DOCUMENT RESUME

ED 146 899 IR 005 280

AUTHOR	Wescourt, Keith T.; And Others
TITLE	Knowledge-Based CAI: CINS for Individualized Curriculum Sequencing. Final Technical Report No. 290.
INSTITUTION	Stanford Univ., Calif. Inst. for Mathematical Studies in Social Science.
SPONS AGENCY	Office of Naval Research, Washington, D.C. Personnel and Training Branch.
PUB DATE	3 Oct 77
CONTRACT	N00014-76-C-0615
NOTE	129p.
EDRS PRICE	MF-$0.83 HC-$7.35 Plus Postage.
DESCRIPTORS	Artificial Intelligence; *Computer Assisted Instruction; *Curriculum Development; Curriculum Planning; Educational Assessment; *Individualized Instruction; Models; Programing; *Sequential Programs; Simulation; Student Centered Curriculum; *Technical Education
IDENTIFIERS	BASIC Instructional Program; *Curriculum Information Network

ABSTRACT

This report describes research on the Curriculum Information Network (CIN) paradigm for computer-assisted instruction (CAI) in technical subjects. The CIN concept was first conceived and implemented in the BASIC Instructional Program (BIP). The primary objective of CIN-based CAI and the BIP project has been to develop procedures for providing each student with an individualized sequence of instruction within the constraints of broader instructional objectives. Although the initial BIP system was for the most part successful in providing individualized problem selection some general weaknesses and specific failures were identified. The present research was concerned with locating problems in BIP's CIN and in developing more robust CIN structures and associated procedures for curriculum sequencing. This research included the implementation of simulation procedure for debugging and preliminary evaluation of CIN-based systems. The simulation was used to examine modifications to BIP's problem-selection procedure. The major effort was the development of a new CIN structure modeled on a semantic network formalism, which was designed to overcome more general limitations of the original CIN structure. The new CIN was implemented in the BIP-II system, and data were collected on the experimental use of this system. (Author)

Documents acquired by ERIC include many informal unpublished materials not available from other sources. ERIC makes every effort to obtain the best copy available. Nevertheless, items of marginal reproducibility are often encountered and this affects the quality of the microfiche and hardcopy reproductions ERIC makes available via the ERIC Document Reproduction Service (EDRS). EDRS is not responsible for the quality of the original document. Reproductions supplied by EDRS are the best that can be made from the original.

U S DEPARTMENT OF HEALTH,
EDUCATION & WELFARE
NATIONAL INSTITUTE OF
EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL NATIONAL INSTITUTE OF EDUCATION POSITION OR POLICY

ED146899

KNOWLEDGE-BASED CAI: CINS FOR INDIVIDUALIZED CURRICULUM SEQUENCING

by

Keith T. Wescourt, Marian Beard, Laura Gould, and Avron Barr

Final Technical Report

Contract No.: N00014-76-C-0615
August 1, 1975 - October 31, 1976

Contractor: Institute for Mathematical Studies in the Social Sciences
Stanford University
Stanford, California 94305

This research was supported by:

Personnel & Training Research Programs
Office of Naval Research (Code 458)
Arlington, Virginia 22217
Contract Authority Number: NR 154-381
Scientific Officer: Dr. Marshall Farr

Approved for public release; distribution unlimited.

Reproduction in whole or in part is permitted
for any purpose of the U. S. Government.

IR 005 280

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Knowledge-based CAI: CINs for Individualized Curriculum Sequencing		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report Aug. 1, 1975-Oct. 31, 1976
		6. PERFORMING ORG. REPORT NUMBER Technical Report No. 290
7. AUTHOR(s) Keith T. Wescourt, Marian Beard, Laura Gould, and Avron Barr		8. CONTRACT OR GRANT NUMBER(s) N00014-76-C-0615
9. PERFORMING ORGANIZATION NAME AND ADDRESS Institute for Mathematical Studies in the Social Sciences, Stanford University, Stanford, California 94305		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61153N RR 042-06; RR 042-06-01 NR 154-381
11. CONTROLLING OFFICE NAME AND ADDRESS Personnel & Training Research Programs Office of Naval Research (Code 458) Arlington, VA 22217		12. REPORT DATE October 3, 1977
		13. NUMBER OF PAGES 120
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

computer-assisted instruction (CAI), instruction control strategy, optimized learning, computer programming, BASIC Instructional Program (BIP), semantic network, artificial intelligence (AI), student model, inference procedures, Curriculum Information Network (CIN), simulation, educational evaluation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes research on the Curriculum Information Network (CIN) paradigm for computer-assisted instruction (CAI) in technical subjects. The CIN concept was first conceived and implemented in the BASIC Instructional Program (BIP) (Barr, Beard, & Atkinson, 1975, 1976). The primary objective of CIN-based CAI and the BIP project has been to develop procedures for providing each student with an individualized sequence of instruction within the

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

constraints of broader instructional objectives. Although the initial BIP system was for the most part successful in providing individualized problem selection, some general weaknesses and specific failures were identified. The present research was concerned with locating problems in BIP's CIN and in developing more robust CIN structures and associated procedures for curriculum sequencing. This research included the implementation of a simulation procedure for debugging and preliminary evaluation of CIN-based systems. The simulation was used to examine modifications to BIP's problem-selection procedure. The major effort was the development of a new CIN structure modeled on a semantic network formalism, which was designed to overcome more general limitations of the original CIN structure. The new CIN was implemented in the BIP-II system, and data were collected on the experimental use of this system.

Preface

This report describes research on the Curriculum Information Network (CIN) paradigm for computer-assisted instruction (CAI) in technical subjects. The CIN concept was first conceived and implemented in the BASIC Instructional Program (BIP) (Barr, Beard, & Atkinson, 1975, 1976).[1] The primary objective of CIN-based CAI and the BIP project has been to develop procedures for providing each student with an individualized sequence of instruction within the constraints of broader instructional objectives. Although the initial BIP system was for the most part successful in providing individualized problem selection, some general weaknesses and specific failures were identified. The present research was concerned with locating problems in BIP'S CIN and in developing more robust CIN structures and associated procedures for curriculum sequencing. This research included the implementation of a simulation procedure for debugging and preliminary evaluation of CIN-based systems. The simulation was used to examine modifications to BIP's problem-selection procedure. The major effort was the development a new CIN structure modeled on a semantic network formalism, which was designed to overcome more general limitations of the original CIN structure. The new CIN was implemented in the BIP-II system, and data were collected on the experimental use of this system.

---

[1]The earlier research was supported jointly by the Office of Naval Research and the Advanced Research Projects Agency. Subsequent support was also provided by the Navy Personnel Research and Development Center, San Diego.

Plan of this report

The organization of this report is as follows. Section I describes the CIN paradigm in relation to other contemporary CAI research on individualized instruction. Section II summarizes the earlier research with a CIN in the BIP system and introduces the problems we sought to understand in the present research. Our analysis of existing and potential techniques-- in particular, simulation methods-- for developing and evaluating knowledge-based CAI systems (such as the CIN-based BIP system), is presented in Section III. Section IV describes our development and use of an automated simulation procedure for testing and evaluating modifications to the BIP problem-selection procedure. Section V describes our development of a CIN structure modeled on a semantic network formalism and its implementation in the BIP-II system, and presents preliminary data on the experimental use of BIP-II. In Section VI, we give our general conclusions on the research program.

Acknowledgement

We are pleased to recognize here the contributions of Mary Dageforde, who helped to implement the BIP-II system, and Alex Strong, who helped to analyze data from the simulation experiments described in Section IV. Dr. Marshall Farr, the Scientific Officer for this research contract, provided guidance through his questions and comments during the course of the project.

I. Individualization in CAI

Goals and progress

Much current research on CAI has stressed the development of systems that can emulate a subset of the intelligent behavior displayed by human tutors. Among the capabilities that have been investigated are:

1) interactive dialog conducted in some reasonable subset of English

2) evaluating student answers by "understanding" them in terms of the subject matter, rather than by simply comparing them to the course author's prepared list of expected right and wrong responses

3) error correction or problem-solving assistance tailored to each instructional scenario

4) dynamic decisions about what and how to teach based on the student's previous interactions.

Significant advances have been made toward representing the knowledge underlying such capabilities and toward implementing them in prototype CAI systems during the past decade. The student-machine interface has been broadened to use English (Brown & Burton, 1975; Carbonell, 1970; Collins & Grignetti, 1975), computer-generated audio (Atkinson, Fletcher, Lindsay, Campbell, & Barr, 1973; Sanders, Benbassat, & Smith, 1976; Van Campen, 1970), voice recognition (Danforth, Rogosa, & Suppes, 1974), and graphics (Bork, 1975). Answer checking and analysis has been investigated using several different approaches suitable for different subject domains: proof checkers (Goldberg, 1973; Smith, Graves, Blaine, & Marinov, 1975), a REDUCE-like algebraic simplifier (Kimball, 1973), solution synthesis systems (Brown & Burton, 1975; Brown, Burton,

Hausmann, Goldstein, Huggins, & Miller, 1977; Brown, Burton, Miller, deKleer, Purcell, Hausman, & Bobrow, 1975; Carr & Goldstein, 1977), and heuristic pattern-analysis procedures (Collins, 1977; Goldstein, 1975; Ruth, 1974). One course under development at IMSSS at Stanford uses a natural language parser, an algebraic simplifier, and a sophisticated proof checker to converse with the student about set theoretic proofs in informal English (Smith & Blaine, 1976).

Since its inception, CAI has had the goal of providing "individualized" instruction, one aspect of which is sequencing the curriculum material optimally for each student. Some early systems applied elements of mathematical learning and decision theories (Groen & Atkinson, 1966; Suppes & Morningstar, 1972). These methods have proved successful for individualizing simple drill-and-practice in elementary mathematics and language arts, but are not extensible to more complex subject domains (e.g., Wollmer & Bond, 1975; see Brown, et al., 1975 for a critique). Our research in CAI has focused on the issue of the "representation" of the subject domain (which is also a fundamental concern of current research in cognitive psychology and artificial intelligence). The goal has been to provide a representation of the subject matter that is sufficient for individualized presentation of a curriculum, while maintaining the advantages for student motivation inherent in a curriculum designed by human authors. To this end, we have developed a representation called a Curriculum Information Network (CIN), which may be best introduced by examining other approaches taken toward individualization of curriculum presentation in CAI.

Curriculum-based branching

The most common style of CAI courseware now being written, which we will call "curriculum-based branching CAI," is an automated presentation of a curriculum written by a human author. The author, knowledgeable in the subject matter, has an implicit understanding of the dependency of one concept on another, and a plan for how and when to teach each one. His organization of the concepts results in an integrated curriculum, presented in a sequence he considers to be optimal in some sense for his model of his students' existing knowledge and learning abilities. This structure resembles that of a textbook, established in advance of interaction with the student, but potentially superior to a textbook in that the author specifies branching points in the curriculum, providing for a degree of individualization. The subdivisions of the curriculum and the branching criteria constitute the representation of the subject matter in this traditional CAI style.

Figure 1 illustrates the basic mechanism of individualization in curriculum-based branching CAI. The instructor-supplied curriculum consists of lessons, exercises, multiple-choice questions, tests, problems, or problem-forms (to be filled in by the instructional program). A record of each curriculum element a student sees and his performance on it is kept in a student history. The student starts with a specified element. Problem-selection rules, which use the information in the history, are used to select subsequent curriculum elements. For example:

```
If correct then go to problem 8, otherwise go to problem 6.
If answer is 9 instead of 6 then student is wrong
      and go to exercise 13.
If percent correct < 75 then go to review lesson III-R1.
If percent correct > 90 then skip next lesson.
```

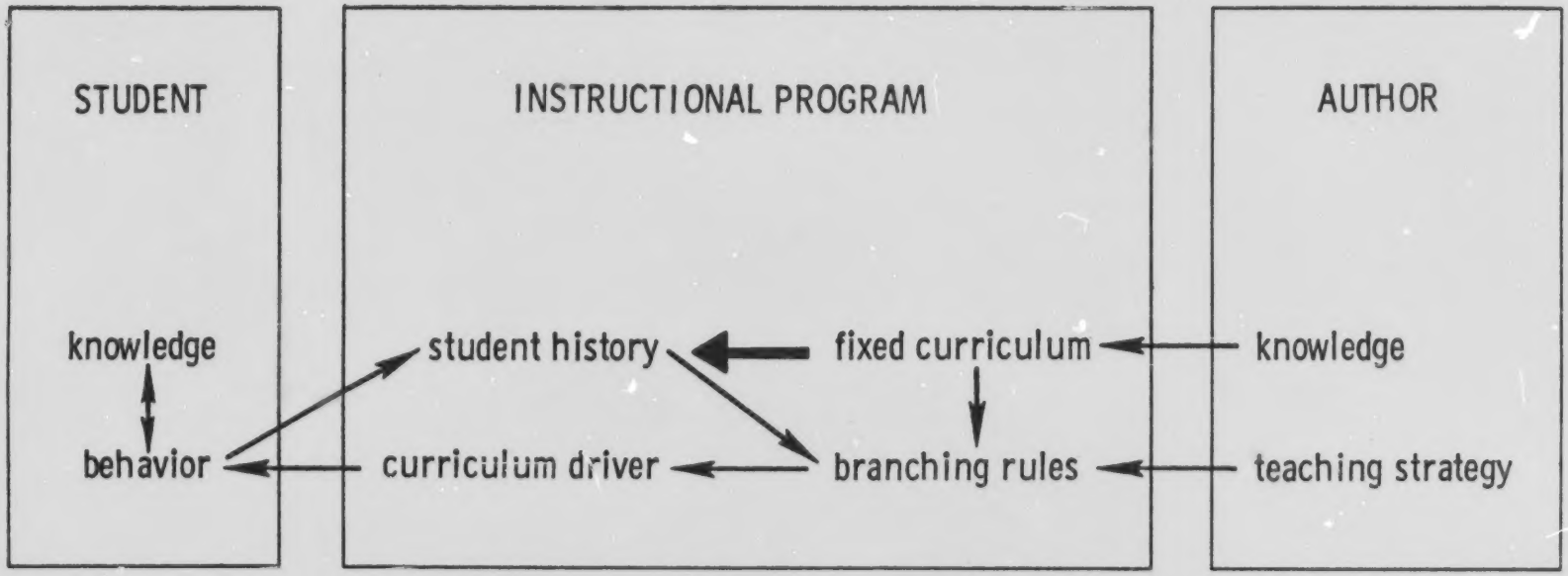

Figure 1. Organization of curriculum-based branching CAI systems.

The crucial point here is that the branching rules are defined in terms of the curriculum elements themselves, which is the only aspect of the subject matter that the program "knows about." These rules can be built into the structure of the curriculum (e.g., as standard decision procedures to be followed at the end of each lesson), or supplied explicitly (e.g., within particular problems, where certain responses can be specially treated); the most typical case is some combination of both. In most cases, however, the net effect of curriculum-based branching for individualization has been only that bright students are allowed to detour around blocks of material, or that less competent students are given remedial lessons.

An example of curriculum-based branching CAI is the AID course developed at Stanford (Friend, 1973). This is a large scale course, with a hundred hours of curriculum material to teach the AID programming language (Algebraic Interpretive Dialogue) at the introductory undergraduate level. The course has been used with success in colleges and junior colleges as a introduction to computer programming (Friend, 1975; Beard, Lorton, Searle, & Atkinson, 1973). It is a linear, "lesson-oriented" CAI program that uses prespecified branching criteria like those described above. One limitation of the course is that it does not monitor the problem-solving activity itself. After working through lesson segments on such topics as syntax and expressions, the student is assigned a problem to solve in AID. It is necessary for him to leave the instructional program, call up a separate AID interpreter, perform the required programming task, and return to the instructional program with an answer. As he develops his program directly with the AID interpreter, his only source of assistance is the minimally informative error messages it provides.

More importantly, individualization within the AID course is inadequate because of the linear organization of its curriculum. The curriculum consists of a large set of ordered lessons, reviews, and tests, and a student's progress from one segment to the next is determined by his score on the previous segment. A high score leads to an "extra credit" lesson on the same concepts, while a low score is followed by a review lesson. This individualization scheme, based on total lesson scores, is reasonably effective in providing instruction and programming practice, but since it deals with rather large segments of the curriculum, the precision with which it can respond to different students' difficulties with specific concepts is minimal. When allowed to select a lesson, students almost invariably choose to proceed to the next (numbered) lesson or to review just-completed material. For example, even if a student recognizes that he is confused about initializing and incrementing, his only choice is to request the review lesson on loops. Because of the AID course's limited ability to characterize individual students' knowledge of specific concepts, and its inability to relate that knowledge to the curriculum beyond determining a ratio of problems correct to problems attempted, all students cover the same concepts in roughly the same order, with slight differences in the amount of review (Beard et al., 1973).

Generative CAI

At the opposite pole of CAI structure are "generative" systems, which do not use an author-written curriculum at all. Instead, problem statements and solutions are generated by retrieving information from a sufficient, internal representation of the knowledge underlying the subject domain.

An important goal of generative CAI is to incorporate the instructor's knowledge of the subject domain into the instructional program itself (see Figure 2). The program has a database containing all the "facts" the student is to learn, as well as the reasoning procedures that the instructor uses to manipulate those facts. It also has pedagogical knowledge of how to present facts that the student doesn't know in a way that will facilitate learning. Its student model is a representation of what the student knows, not just what his behavior has been on specific previous exercises.

The SCHOLAR system. There are several examples of excellent generative CAI programs in various subject areas (e.g., Brown & Burton, 1975; Brown, et al., 1975; Collins & Grignetti, 1975). For subjects such as geography, where the student's goal is to master a set of facts, the SCHOLAR system (Carbonell, 1970; Collins & Grignetti, 1975) is an advanced prototype. SCHOLAR uses question-and-answer construction algorithms to engage the student in a dialog about a subject domain. Carbonell's original goal was to build a program capable of generating questions and answers in any subject area in which the information represented was ill-defined verbal knowledge rather than a more well-structured subject like mathematics. Subsequent research with SCHOLAR built on this original idea, first expanding the question-and-answer generating capabilities into other subject areas such as on-line text editing (which requires that students learn procedures as well as facts) and then focusing on the reasoning skills used to answer questions given incomplete knowledge. With respect to individualizing instruction, the concern of the SCHOLAR system has been, generally speaking, to respond more appropriately to the student in the

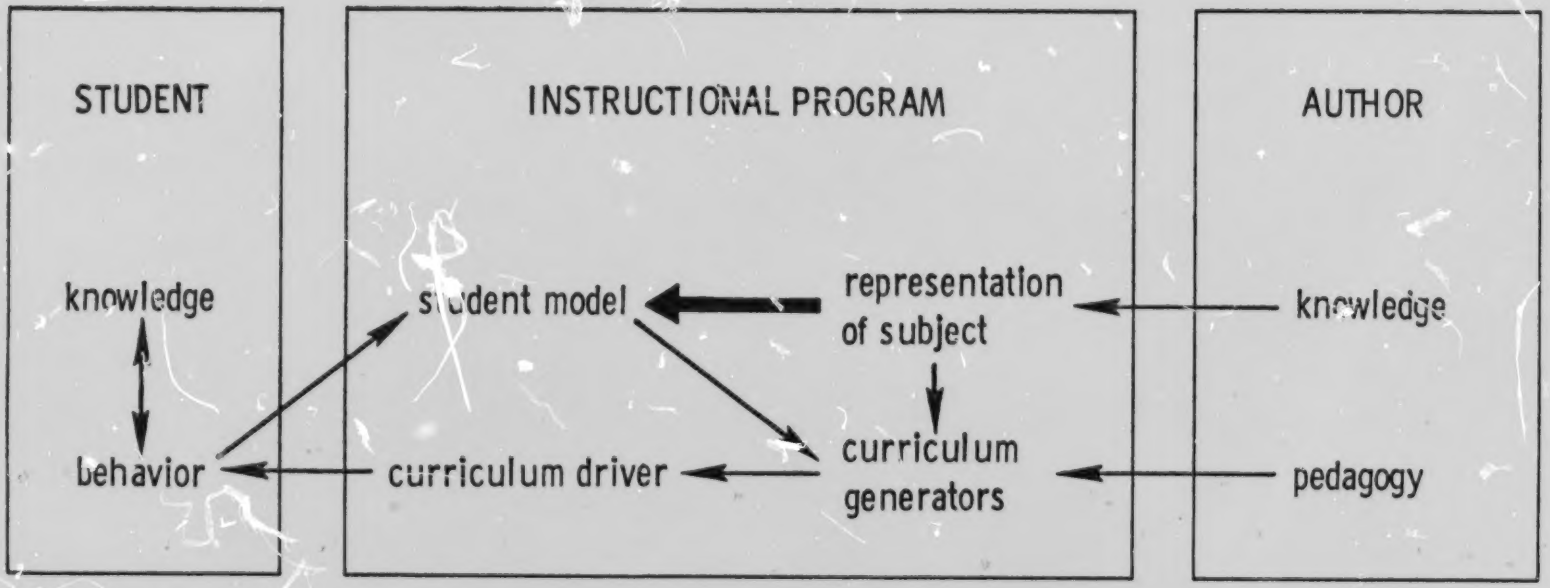

Figure 2. Organization of generative CAI systems.

immediate situation; that is, the aim has been to enable the system to generate examples based on the student's most recent error, or to generate a next question related to his most recent hypothesis about a fact. In both cases, the system's actions are determined by a model of the student's familiarity with all the facts in the database and the learning goals inferred for his most recent interaction..

However, SCHOLAR does not systematically individualize the presentation of curriculum on a larger scale. The semantic network it uses to represent the subject domain is organized as an outline of topics and subtopics, each with an "I-tag" indicating its importance. In the version of SCHOLAR that tutored geography, topic selection worked as follows: Within time constraints, the program discussed the information under an arbitrary topic down to a pre-specified level of importance. When the allotted time expired, the program backed up to another high-level topic and began again to ask questions, provide related information, and give review down through the subtopics in the order of their importance. The version of SCHOLAR that taught the use of the on-line editor proceeded through a set of lessons; hints and error correction were generated dynamically in response to the student's input, but the overall path through the "curriculum" was fixed.

Although the later work with SCHOLAR moved beyond the teaching of isolated facts,, the dialog between the program and the student was still characterized by episodes dealing with a single question at a time. The NLS-SCHOLAR system (Grignetti, Hausmann, & Gould, 1975) allowed the student access to the text editor itself, but the hands-on problem solving involved only a very limited sequence of editing changes. In general, SCHOLAR did not deal with more complex

problem-solving episodes requiring the integration of extended factual and procedural knowledge.

The MALT system. The MALT system (Koffman & Blount, 1975) for teaching introductory programming in machine language illustrates the issues involved in producing generative courseware in subjects centered around complex problem solving. MALT uses a set of programming primitives to generate programming tasks (by combining primitives) which it can both present to the student, in English, and solve with a program, since it can solve all of the primitive tasks. One advantage of this approach is that the system can generate and solve [illegible] variety of problems. Another, perhaps more important in our view, is the system's ability to present increasingly difficult problems as a function of each student's competence and prior experience with the primitives. Koffman (1972) describes his "intelligent CAI monitor" as:

> . . . centered around the use of a student model (summary of a student's past performance) and a concept tree, which indicates the pre-requisite structure of the course. As the system gains experience with a particular student, it updates his model and establishes to what extent he prefers to advance quickly to new material or build a solid foundation before going on. Based on its knowledge of the student and his past performance, it decides at which plateau of the concept tree the student should be working. All concepts on this plateau are then evaluated with respect to factors such as recency of use, change in state of knowledge during last interaction, current state of knowledge, tendency to increase or decrease his state of knowledge, and relevance to other course concepts. The highest scoring concept is selected, a problem suitable for his experience level is generated, and its solution is monitored.

The disadvantages of the MALT system are that it requires the student to follow its own sequence of primitive steps in solving the

problem, and that its problem statements are uninteresting and unmotivating compared to those which a human tutor would develop. For example:

> Your problem is to write a program which will:
> Read in 10 (octal) 1-digit numbers and store their values
> starting in register 205.
>
> Here are the sub-problems for the 1st line:
>
> (1) Initialize a pointer to register 205.
> (2) Initialize a counter with the value of -10 (octal).
> (3) Read a digit and mask out the left 9 bits.
> (4) Store it away using the pointer.
> (5) Update the pointer.
> (6) Update the counter and if it's not zero, jump back to start of loop.

Thus, although the problem-generation process enables the MALT system to individualize the sequence of instruction, it sacrifices the depth and richness of content that makes problems interesting to students.

To summarize, the advantages for individualization of generative CAI over curriculum-based branching CAI are considerable: the generative program can provide tutorial instruction in specific areas relevant to the student's needs. All decisions about what material to present can be made dynamically, based on the individual student's overall progress with the subject matter, rather than on his responses at pre-determined choice points in an otherwise fixed sequence. Ideally, the program embodies the same information that makes its human author a subject matter expert, and this information can be made available to the student much more flexibly than is possible in curriculum-based branching CAI. The major disadvantage of generative CAI, at least for technical subjects such as computer programming, is that generated <u>questions</u> limit the student's hands-on interactions with the subject matter, while generated <u>problems</u> tend to be unmotivating to students.

The Curriculum Information Network

The approach we have developed using the Curriculum Information Network adapts the techniques for individualization of problem selection used in generative CAI to problems written by human authors. Thus, it attempts to gain the advantages of both generative and more traditional approaches to CAI.

In technical subjects, development of skills requires the integration of facts, not just their memorization, and the organization of instructional material is crucial for effective instruction. As the curriculum becomes more complex, with each curriculum element involving the interrelations of many facts, the author's ability to present it in a way that facilitates assimilation and integration becomes more important. At the same time, a history of performance on specific lessons, questions or problems per se becomes a less adequate model of the student's acquisition of knowledge. The CIN is a means for representing the complex knowledge underlying problems in technical curricula and for modeling the learning of that knowledge.

The CIN provides the instructional program with an explicit representation of the structure of an author-written curriculum. It depicts the relationships between problems and the concepts that they involve which, presumably, the author would use implicitly in determining "branching" schemes for sequencing the problems. Using the CIN, student learning can be modeled in terms of acquisition of the concepts, not just a history of right and wrong responses on the problems. The CIN includes a description of each author-written problem in terms of a subset of domain-specific skills needed to achieve a solution. The instructional program can monitor the student's use of

these skills, and choose the next problem with an appropriate group of new skills. As the student completes or fails problems, the CIN serves as a model of his state of knowledge, since it has an estimate of his ability in the relevant skills, not just his performance on the problems he has completed. Branching decisions are based on this model instead of being determined simply by the student's success/failure history on the problems he has completed, as shown in Figure 3.

In curriculum-based branching, simple problems may focus on one particular skill which, by itself, the student may have mastered. On the other hand, complicated problems may involve a large number of different skills, some of which are beyond the student's ability to learn when the problem is presented. Neither of these experiences is likely to result in significant progress toward learning the subject.

Generative programs enable more productive learning episodes by creating problems that focus on skills a student has demonstrated difficulty with or those new skills which extend his prior learning. But, as we have noted, the program-generated problems are typically very boring, resembling mechanical exercises rather than challenging, interesting tasks. The CIN approach also selects problems involving appropriate skills, reducing the likelihood that the student will become bored or frustrated by the difficulty of his task. However, in addition, the CIN provides the capability to present more motivating human-authored problems.

The following is an example of a programming problem taken from our CAI course in programming, which will be described in detail in the next section:

On the first day of Christmas, someone's true

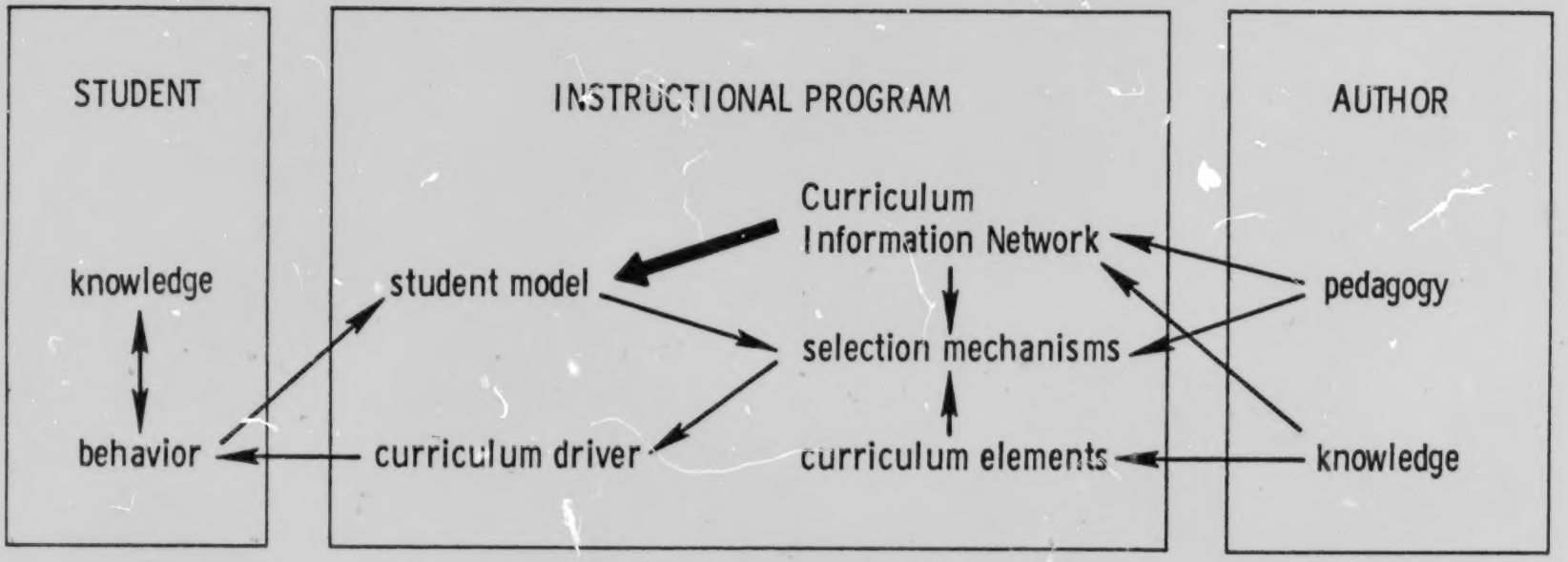

Figure 3. Organization of CIN-based CAI systems.

love sent him/her a partridge in a pear tree (one gift on the first day). On the second day, the true love sent two turtle doves in addition to another partridge (three gifts on the second day). This continued through the 12th day, when the true love sent 12 lords, 11 ladies, 10 drummers, . . . all the way to yet another partridge.

Write a program that computes and prints the number of gifts sent on that twelfth day. (This is not the same as the total number of gifts sent throughout all 12 days -- just the number sent on that single 12th day.)

The skills that describe this task are:

Initialize numeric variable (not counter) to literal value
FOR . NEXT loop with literal as final value
Accumulate successive values into numeric variable
Multiple print: string literal, numeric variable

Since problems are selected on the basis of the student's performance on the skills underlying the curriculum, this problem might be presented either when the student is ready to learn about FOR . NEXT loops that accumulate a sum, or after he has had difficulty with such skills in a different problem, and therefore needs more work on those skills.

Computer-assisted instruction has long promised to present an individualized sequence of curriculum material, but in most cases this has meant only that bright students are allowed to detour around blocks of curriculum, or that less competent students are given sets of remedial exercises. By describing the curriculum in terms of the skills on which the student should demonstrate competence, and by selecting problems on the basis of individual achievement and difficulties on those skills, more meaningful individualization can be attained. We have explored the CIN approach for CAI in computer programming, but it should be applicable in many other subject areas that involve identifiable skills and that require the student to use those skills in

different contexts. The next section reviews our earlier development of a specific Curriculum Information Network and the procedures used to select problems adaptively, and summarizes the results that motivated the further research on CIN-based CAI conducted under the present contract.

II. The BIP-I CAI Program

Overview

This section describes our initial implementation of a Curriculum Information Network in a fully operational CAI course. Our experience over the past three years with the BASIC Instructional Program (BIP) has given us insights into the power and limitations of the CIN approach. The development of the BIP system has been supported by the Office of Naval Research, the Advanced Research Projects Agency, and the Navy Personnel Research and Development Center. BIP-I, the version of the program operational prior to the present contract research, is fully described by Barr, Beard, and Atkinson (1975, 1976).

BIP is designed to introduce students to programming in the BASIC language, almost exclusively through guided hands-on practice in writing and running programs. Figure 4 illustrates the relationships among the parts of the BIP system. Using the information in the CIN and the student model, the task-selection procedures present the student with a problem ("task") to solve, typically of the form "Write a program that . . ." As he types his program, the interpreter presents specially-designed instructional messages when errors occur. The student also has access to hints (both graphic and text) and debugging facilities, and he may execute the stored "model solution" at any time to see how his own program is expected to behave. The solution checker evaluates his program by comparing its output to that of the model solution; if his program is not acceptable, he may choose either to leave the task at that time or to continue working on his program. When

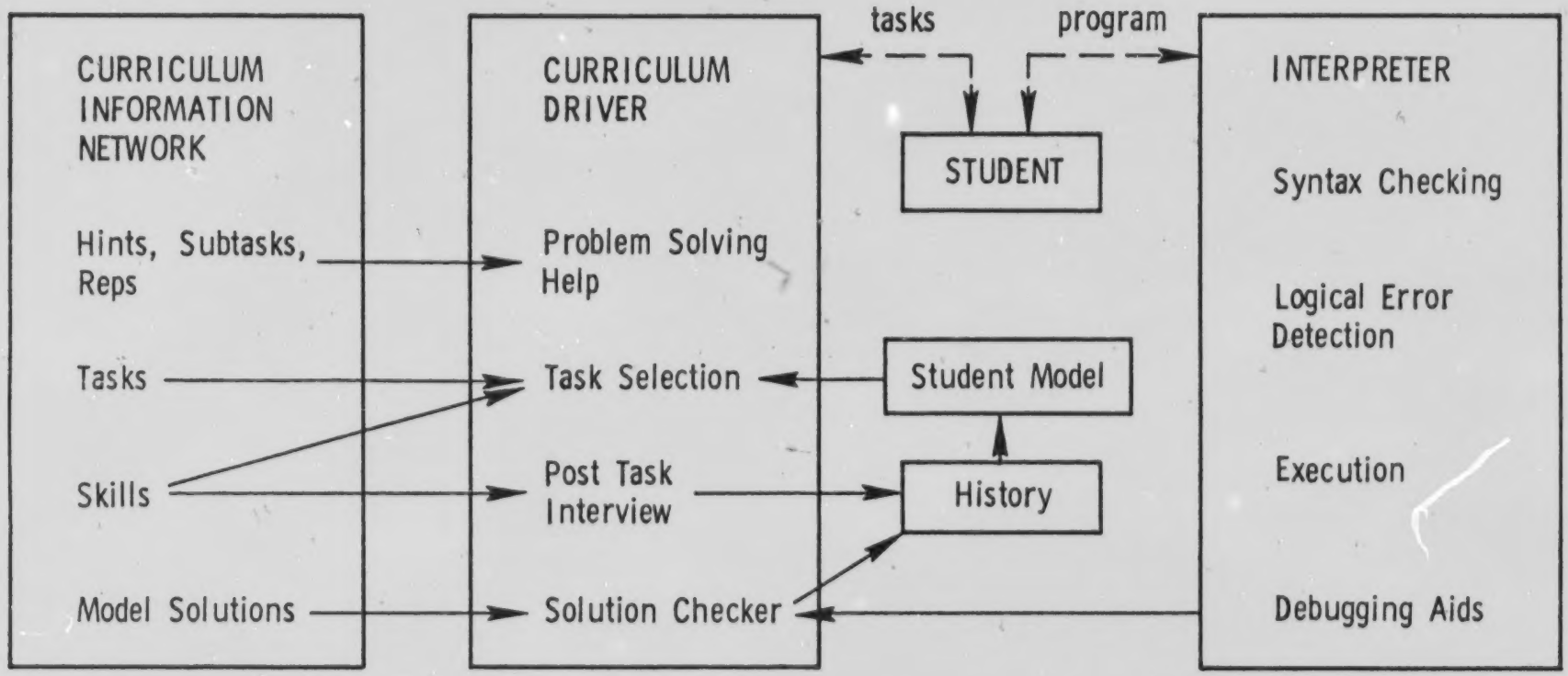

Figure 4. Organization of the BIP system.

he leaves, either by "quitting" at this point or by successfully completing an acceptable program, the "Post Task Interview" (PTI) asks him to evaluate his own competence on the skills involved in the task. The student model is updated to reflect the student's demonstrated competence with the skills in the task and his responses to the PTI.

Dynamic task selection in BIP-I

The distinctive feature of the CIN-based system is the existence of the tasks as an unstructured pool of curriculum elements available for presentation to the student; the actual sequence in which he sees the tasks depends on a stored "advance" strategy only when a dynamic decision process fails. The tasks in BIP represent widely varying degrees of difficulty, and it is certainly possible to order them in any one of a number of pedagogically reasonable fixed sequences. This kind of sequencing resembles the order of chapters in a textbook, a reasonable path for some hypothetical "average" student. The purpose of the CIN and its associated student model and task-selection procedure is to use the information from a particular student's interaction to order the presentation of tasks; the system is designed to respond to individual differences in ways that are much too complex for the author to have anticipated, much less specified fully beforehand for all students.

The most important part of the CIN for purposes of task selection (as opposed to helping the student solve the problem, for example) is the relationship between each task and its procedural skills. The skills presently defined in BIP are rather specific descriptions of programming constructs (as opposed to more general

problem-solving methods), such as "print a string literal" or "initialize a counter variable." Each task is described by the set of skills that are needed to write an acceptable solution program, and student learning is modeled with respect to the skills, and not the tasks per se. This "skill history" is crucial to determining the most appropriate next task for a student at any point in his experience. The skills represent knowledge about (or mastery of) aspects of programming itself, and a given task is selected because it embodies programming knowledge that requires remediation or is ready to be learned, as determined from the model of the student's existing knowledge.

Although the skills are, for the most part, defined in terms of syntactic constructs, BIP-I task selection does not reflect the student's knowledge of syntax, but instead depends on his knowledge of the semantics and pragmatics for using the skill appropriately. All purely syntactic errors are detected immediately by the BIP-I interpreter, which responds with explicit feedback describing the error and illustrating syntactically correct examples of the construct. These syntactic errors do not affect the model of the student maintained by BIP-I. This model is affected by logical errors which allow the student's program to run, but which cause it to produce incorrect results. Many of these errors can be associated with semantically or pragmatically inappropriate use of the syntactic constructs described by BIP-I skills.

In BIP-I, the skills are grouped into sixteen non-overlapping sets called techniques, such as simple printing, assignment, and conditional branching. (The skills and the technique groups are listed in Appendix A.) The techniques themselves are linearly ordered according

to judgments about the relative complexity of the skills they contain. The technique ordering is used in BIP-I as a constraint on the order in which major concepts are introduced and as a scale for determining whether problems available for remediating particular skills are appropriate to the student's overall progress.

The BIP-I task-selection procedure based on techniques works as follows: starting at the lowest (least complex) technique, all the skills grouped at that level are compared to the student model; if any skills are considered to "need further work," a task using some of those skills is sought. (If no such "needs-work" skills are found, the skills in the next higher technique group are compared to the student model and the process continues.) If a task is found that uses some skills needing work, and that does not require any skills at a higher level, it is selected and the process ends. If no task can be found at a suitably low level, the process moves to the next higher technique and re-applies the criteria. Thus, the selection mechanism in BIP-I relies on the technique hierarchy for its overall sequencing: skills at low levels are sought before those at higher levels, which, in general, simply means that easier tasks precede more difficult tasks. Within this framework, however, the sequences of tasks selected for different students has proven quite variable.

A skill is considered to be "mastered" if a student has completed a task using that skill successfully, both by passing the solution checker and by indicating in the PTI that he understood the use of the skill in the context of the task. A student who has no difficulty with BIP-I's curriculum, always passing the solution checker and indicating his confidence in his own understanding, follows a

completely predictable path through the techniques. More typically, students have trouble with various concepts and programming structures, and therefore follow widely divergent paths, since BIP-I's evolving model of their knowledge indicates different patterns of mastery of the skills. Some students remain at a given technique level for longer periods of time, receiving more work on certain skills; some drop down to lower levels for specific remediation on other skills, etc. The variety of resulting sequences resembles the variety of experience provided by a strictly generative CAI course, since progress is measured by success within the representation of programming knowledge rather than on a set of lessons.

Evaluation of BIP-I

Although use of BIP-I demonstrated that its task selection is responsive to individual students' needs, it revealed weaknesses in its specific CIN implementation. We will discuss some of the observed strengths of the BIP-I system and some of the drawbacks which motivated the research undertaken in the present contract (see Sections IV and V).

One useful feature of BIP-I is the feedback it generates about its task-selection process. The system maintains a record of failures in seeking tasks with particular characteristics. If no task can be found that requires some "needs work" skills without requiring any skills at higher levels, this failure is recorded as a "hole" in the curriculum. The information describing the hole includes the skills that were being sought, the technique level that the process had reached, and the reason for the failure -- either no task with "needs work" skills or none with skills at suitably low levels. About 20 tasks

were added in one major curriculum revision of BIP-I and another, based almost entirely on the holes recorded since that time, was carried out in the context of the present research leading to the development of the BIP-II system. By comparison, to determine the content weaknesses of his curriculum in traditional curriculum-based branching CAI, the author must assemble and analyze a variety of indirect data such as test performance. The record that the CIN-based system makes of its failures gives immediate, direct, automatically-generated data indicating precisely those areas in which the curriculum and its sequencing are deficient.

Other system data provide direct information on the nature of both student and program performance. The student model is itself a reflection of the student's progress not through the tasks but through the specific aspects of programming represented in the CIN. The patterns of concentration on certain skills (and the lack of emphasis on others) indicates those areas of the subject matter that may be receiving too much (or too little) attention. Such weaknesses indicate the need for changes in either the content of the curriculum, or the technique groupings, or the task-selection process. All of these approaches, singly and in combination, were used during the evolution of BIP-I to improve the balance of skills presented. We emphasize that the CIN-based design allows experimentation with different mechanisms for selecting among tasks without changing the tasks themselves. Again, the contrast with fixed-branching CAI is strong: Since such curricula are conceived of as a whole, it is almost impossible to change the way in which the problems are presented without changing the whole structure (usually lessons) in which they are imbedded.

During the autumn of 1976, the BIP-I system was made available to the U.S. Naval Academy for an operational evaluation in a Navy setting. Thorough analysis of all aspects of the operation is not within the scope of this report, but we draw on examples from the data to illustrate a few specific points about the strengths and weaknesses of BIP-I's CIN. In general, the midshipmen's experience with BIP-I was favorable; specifically with respect to the sequence of tasks selected, the CIN seems to have provided considerable individualization in response to students' different rates of progress.

One measure of the "goodness" of a sequence of tasks is the degree to which students progressed smoothly through the technique levels, which generally reflect the increasing complexity and difficulty of the tasks. Ideally, the sequence should involve no large jumps, either upward or downward, in complexity. Among the 534 tasks selected by BIP's mechanism (i.e., not specifically requested by the student himself), there occurred 71 instances in which the technique level changed by more than one, either upward or downward. (Only five of the 16 students experienced more than five such "breaks" in the progression.) The causes of the "breaks" include:

1) simple failure of the process to perform in a pedagogically correct way -- actual design problems which will be discussed further in this section

2) student requests for more work on skills at low levels -- requests which are always honored if a sufficiently easy task is available

3) mastery of certain skills in the context of tasks chosen explicitly by the student. (If a student chooses to select a few tasks by name, the next task selected by BIP-I is often at a technique level three or four higher than that of the BIP-I selection that preceded the student's self-guided sequence.)

Thus, because of the "breaks" that were caused by students' requests for specific tasks or for more work on specific skills, the "failure" rate of our CIN implementation due entirely to weaknesses in the design is actually lower than 71/534. This ratio indicates the general viability of CIN-based task selection in BIP-I. (Section III more thoroughly examines the issues involved in evaluating the effectiveness of CIN-based task selection.)

The major weakness of task selection in BIP-I can be traced to the use of the technique structure as a governing constraint. The skills at a given technique level are not necessarily analogous to each other in the sense of dealing with similar concepts or similar programming semantics -- they are just judged to be similarly difficult to use. The sequence of tasks that results from the technique-based task-selection process occasionally appears to be arbitrary with respect to the content of the problems, even though the progression of difficulty appears reasonable. The techniques also add little to the ability to make useful inferences about the different contexts in which a given skill might appear, differences that might contribute to a student's difficulty with a supposedly well-learned skill. The technique groupings were intended to provide an overall guide for the sequencing of tasks, and they were successful enough as a first approximation in attaining this goal. However, the technique structure of BIP-I does not specify the relationships among skills precisely enough to capture the complexity of the programming knowledge taught by the curriculum. There are a number of specific symptoms of this general weakness which the following cases of pedagogical inadequacy illustrate.

One student's record shows a sequence of tasks at technique

levels 10, 11, 13, 15, then suddenly dropping back to a task at Technique 6. More important than the drop in numbers was the difference between the two adjacent tasks. The student had successfully completed task BACKARRAY, which requires a program that obtains an array of words from the user, and prints the array in backwards order. The next task selected by BIP was INPUTSUM, which requires a program that gets just two numbers from the user and prints their sum. Obviously, INPUTSUM is too easy for a student who successfully handled BACKARRAY. BIP selected the easier task for the following reason: The student had quit a difficult task at Technique 13 (choosing to leave after failing the solution checker); one of the skills in that task was 29, which appears in a much lower technique and which the student had used successfully in one earlier task. After a quit, the counter representing learning of each of the skills in the task is decremented. The student chose the BACKARRAY task himself, and passed it, but since it did not require Skill 29, that skill's counter was still at zero, indicating that he needed more work on it. Thus, when the task-selection process climbed through the techniques, it identified 29 as a skill to be sought in the next task, and then found INPUTSUM, which requires that skill, at Technique 6.

This illustration typifies the more severe failures of the task-selection procedures to locate an appropriate next task. Students who have progressed very rapidly up to the time that they quit a difficult task are particularly vulnerable to this kind of "drop," since they are much more likely than slower students to have seen many skills only once previously. The student model of BIP-I often does not accurately reflect the student's knowledge of the skills. Also, the

usually successful principle of beginning the search for "needs work" skills at the lowest technique has important exceptions.

A second illustration illuminates another problem with the CIN implementation of BIP-I. A student quit task PITCHER, a fairly difficult task at Technique 12, and was next presented with SREAD, a much easier task at Technique 5. (He had previously succeeded with a few tasks at higher levels, so SREAD definitely appears to have been too easy.) The cause of this drastic drop was, again, a single skill (16) which, when decremented after he quit PITCHER, was considered to need more work. Skill 16 occurs at a fairly low technique, and is required in task SREAD, which was therefore presented next. One interesting feature of the PITCHER - SREAD sequence is that only two of the skills required in PITCHER were in the MUST set of "needs work" skills when PITCHER was selected; yet it was a skill <u>not</u> in the MUST set (namely 16) whose decrementing led to the too-easy next task. This anomaly occurs because the MUST set is established when a new task is requested, not when the current task is completed. Thus, in this case, the skills in the MUST set at the time PITCHER was selected were ignored when SREAD was selected (because they all appear at higher levels). Establishing the MUST set after the completion of each task, so that the requirements for the next task would be more similar to that last task than is currently the case, is a possible solution. However, a potential side-effect is a restriction on the range of different skills the student is required to use, especially in cases of failure when more drastic variations might be most pedagogically effective.

Other issues related to using a CIN for task selection and to modeling student learning arose in our evaluation of BIP-I. For

example, how should the model reflect a student's difficulty with a task, or his choosing to leave the task without even attempting to write a solution program? BIP-I almost always gives the student the benefit of the doubt. It ignores "difficulty" -- tremendous amount of time spent in a task, repeated failures to pass the solution checker, etc. If a student writes an acceptable program for a given task, BIP-I assumes that he has learned the material presented in that task, and the counters representing learning of its skills are updated as though he had passed on his first attempt. Similarly, if he chooses to leave the task without yet having failed the solution checker, BIP-I makes no inferences about his mastery or difficulty with the skills involved; their counters are neither incremented nor decremented. Our reasoning is that a student should be allowed to avoid tasks that are not interesting to him. If a student leaves a task, the next task selected should have a very similar group of skills, so we have not worried about students missing significant portions of the curriculum. (Of course, exercising this option can be taken to an extreme, and in some controlled studies we have disabled it.)

Another question deals with the parameters of the task-selection process. Given that a MUST set of skills has been determined, should the system present a task that has as many of those skills as possible (as is the case in BIP-I), or should the proportion of MUST skills be somehow related to the student's overall competence or most recent performance? For example, it might be more reasonable to find the task with only one MUST skill early in the student's experience with the course, or after a succession of "quit" situations. As he gains confidence and competence, the number of allowable MUST skills could

increase, theoretically resulting in an increasing rate of increasing difficulty. A problem with this scheme lies in the relative non-redundancy of the curriculum at the higher levels: Just as PITCHER required only two of the MUST skills, it may often be impossible to find tasks with an arbitrary number of MUST skills (see Section IV); thus, a relatively large curriculum, containing tasks involving many different skill combinations, may be required in order to effectively vary parameters of the task-selection procedure.

These comments reveal some of the complexities and difficulties encountered in designing and using a CIN to individualize task sequencing in BIP-I. The next section presents our analysis of the methods for developing knowledge-based CAI systems and describes our application of them under the present contract to investigate modifications to BIP-I and to create the BIP-II program.

III. CAI Development and Evaluation Techniques

Our work under the current contract to explore further the CIN approach has led us to consider more generally the methods for developing and evaluating prototype knowledge-based CAI systems. In this section, we describe and analyze these methods and their role in modifying BIP-I and implementing BIP-II. There are two sources of data for evaluating a CAI system's performance: operation with real students and simulation. Both have advantages and drawbacks, and the problem is to decide how the two sources can complement one another.

Uses of student data

The statistical analysis of data from field studies (objective measures of learning and subjective ratings) is an accepted method for evaluating the pedagogical effectiveness of instructional systems. In the evaluation of computational correctness and pedagogical adequacy[2] of a CAI system, the primary approach is expert analysis of the interactions between the instructional system and students. The data of interest are records of these interactions, which we call protocols. A protocol is a chronology describing one student's interaction, which for the purpose of evaluation must be detailed enough to allow reconstruction of the system's internal states at each point in time. For convenience, the protocol may explicitly include certain data about system states regularly needed during analysis; for example, BIP-I protocols list those skills judged to need more work each time the

---

[2] Pedagogical adequacy refers to the extent to which the system's behavior is consistent with the pedagogical principles intended by its authors, independent of whether those principles are actually pedagogically effective for a student population.

student requests a new task. Protocols may record interactions at different levels of detail. In a generative question-and-answer dialog system, a protocol might be a verbatim transcript of everything typed by both the student and instructional system (Figure 5). The protocols saved by the BIP-I system are specially formatted blocks of information describing the system's response to each BIP command or line of BASIC code typed by the student (Figure 6).

The complete protocols collected for a representative sample of students provide an essential basis for assessing the correctness and adequacy of a system over the range of instructional situations in which it is intended to function. However, use of a system by real students is too expensive to be practical for evaluating each stage of the many modifications made to an evolving CAI system. In addition, if data are to be collected for the purpose of evaluating a system's pedagogical effectiveness, then all aspects of the system's behavior must be tuned and tested beforehand. Otherwise, program bugs and unexpected inadequate system behavior may interfere with the controlled implementation of the instructional treatments being evaluated. Simulation provides a supplementary source of data for evaluation both before and after the collection of data from students.

Real-time interactive simulation

The simplest type of simulation used for evaluating program correctness and pedagogical adequacy is real-time interactive simulation, which is nothing more than an extension of standard program testing techniques. Interactive simulation is most useful in investigating very specific aspects of system behavior in which the

LET'S BEGIN BY TALKING ABOUT THE LOCATION OF BRAZIL. TRUE OR FALSE? BRAZIL IS IN NORTHEASTERN SOUTH AMERICA.

THAT IS FALSE

NO. BRAZIL IS IN NORTHEASTERN SOUTH AMERICA.

THE EASTERN COAST OF BRAZIL IS THE ATLANTIC. THE WESTERN BORDERING COUNTRIES ARE PERU AND COLOMBIA. THE SOUTHERN BORDERING COUNTRIES ARE URUGUAY AND ARGENTINA.

Because the student missed the question, SCHOLAR presents some related information about the location of Brazil, and then goes on to the next topic.

NEXT WE'RE GOING TO DISCUSS THE TOPOGRAPHY OF BRAZIL. THE PRINCIPAL RIVERS OF BRAZIL ARE----.

THE AMAZON AND THE ORINOCO.

THE AMAZON IS CORRECT. THE ORINOCO IS IN VENEZUELA AND COLOMBIA, NOT IN BRAZIL. THE OTHER PRINCIPAL RIVERS OF BRAZIL ARE THE TOCANTINS, THE SAO FRANCISCO, THE RIO NEGRO, AND THE MADEIRA.

THE SAO FRANCISCO IS IN EASTERN BRAZIL. IT IS IN THE BRAZILIAN HIGHLANDS. THE RIO NEGRO IS IN THE AMAZON JUNGLE. IT IS IN NORTHWESTERN BRAZIL.

Because the student knew about the Amazon, SCHOLAR takes it up as a topic for a while.

CORRECT OR INCORRECT? THE AMAZON IS NAVIGABLE.

POSSIBLY YES

NO. THE AMAZON IS NAVIGABLE.

SCHOLAR misunderstood the "possibly" and treated the answer as wrong.

THE AMAZON IS IN THE AMAZON JUNGLE. IT IS IN BRAZIL AND PERU. THE PRINCIPAL TRIBUTARIES ARE THE RIO NEGRO, THE MADEIRA, THE TOCANTINS, THE TAPAJOZ, THE PURUS, AND THE JURUA.

WHAT IS THE LENGTH OF THE AMAZON?

Figure 5. Dialog between a student and the TUTOR-SCHOLAR system (from Collins and Grignetti, 1977).

IS THE AMAZON A LAKE?

I DON'T KNOW IF THE AMAZON IS A LAKE, BUT IT IS A RIVER.

SCHOLAR found that both lakes and rivers were water, so it couldn't reject the idea.

NOW YOU CAN EITHER ANSWER MY QUESTION OR ASK A NEW ONE.

WHAT IS THE QUESTION?

WHAT IS THE LENGTH OF THE AMAZON?

5000 MILES

NO. THE LENGTH OF THE AMAZON IS 4,000 MILES.

Having exhausted its allotted time to rivers, SCHOLAR went onto the next subtopic under topography of Brazil.

THE PRINCIPAL MOUNTAINS OF BRAZIL ARE ----.

I AM NOT SURE, ARE THE ANDES IN BRAZIL?

IN WHAT YOU SAY, I DON'T RECOGNIZE THE WORDS "I", "AM" AND "SURE". TRY AGAIN.

ARE THE ANDES IN BRAZIL?

NO. THE ANDES ARE NOT IN BRAZIL, BUT IN VENEZUELA, COLOMBIA, ECUADOR, PERU, BOLIVIA, CHILE, AND ARGENTINA.

FEEL FREE TO ANSWER THE QUESTION YOU DIDN'T ANSWER BEFORE.

BRAZIL HAS NO MOUNTAINS.

NO. I DON'T KNOW WHAT YOU MEAN BY NO. THE CORRECT ANSWER IS THE BRAZILIAN HIGHLANDS.

SCHOLAR didn't understand the answer, but proceeded quite reasonably.

THE BRAZILIAN HIGHLANDS ARE IN SOUTHEASTERN BRAZIL. THEY ARE NEAR THE ATLANTIC.

WHAT IS THE PRINCIPAL PLATEAU OF BRAZIL?

The dialogue continued discussing different aspects of Brazil, but we will break here to the beginning of the second pass.

Figure 5 (cont.)

NOW WE WILL GO OVER THE MATERIAL AGAIN. LET'S BEGIN BY TALKING ABOUT THE LOCATION OF BRAZIL.

PLEASE INDICATE IF THE FOLLOWING STATEMENT IS CORRECT OR INCORRECT: BRAZIL IS IN NORTHEASTERN SOUTH AMERICA.

TRUE

THAT'S RIGHT. BRAZIL IS IN NORTHEASTERN SOUTH AMERICA.

Because on this pass the student was correct, SCHOLAR asked questions about the information that was presented on the first pass.

THE EASTERN COAST OF BRAZIL IS ----.

ATLANTIC SEA.

VERY GOOD. THE ATLANTIC IS CORRECT.

THE WESTERN BORDERING COUNTRIES OF BRAZIL ARE ----.

The dialogue continued through the second pass.

Figure 5 (cont.)

```
signon
        experimental subject
        2124    SAM SPADE
        FRIDAY, FEBRUARY 7, 1975 11:58:37PM-PST

                [The student, Sam Spade whose ID number is
                 2124, logs in to BIP]

task
        FRIDAY, FEBRUARY 7, 1975 11:59:10AM-PST
        17
        bip's choice
        technique:  5
        must:  13  14  15  16  30  55  74

                [BIP is asked to select a task.  It chooses
                 task #17, that has its most advanced skill(s)
                 at technique level 5.  The "needs work" set of skills
                 BIP was trying to include in the task it
                 selected was {13, 14, 15, 16, 30, 55, 74}.]

scratch
        FRIDAY, FEBRUARY 7, 1975 12:00:49PM-PST
        *

                [The student clears his program space.]

demo
        FRIDAY, FEBRUARY 7, 1975 12:04:46PM-PST
        17

                [The DEMO is requested: it shows how the
                 student s program should behave like when it
                 runs.  ]

hint
        FRIDAY, FEBRUARY 7, 1975 12:05:34PM-PST
        17
        no hints

                [The student wants BIP to give him a hint;
                 unfortunately there are none for task #17]

model
        FRIDAY, FEBRUARY 7, 1975 12:06:53PM-PST
        17
        10 INPUT X
        *

                [He requests and views the model solution stored
                 for the task.  (He is clearly having problems.)
                 The line "10 INPUT X" is the code he has written
                 to this point.]
```

Figure 6. Section of a BIP protocol recording the interactions of one student across two tasks. Annotations are indented and enclosed in square brackets.

```
syntax error
        99 PRINT END
        1
```

[A syntax error (#1) was detected when he typed the statement "99 PRINT END"-- the error was missing quotemarks around a string literal, but the real mistake is combining 2 statements (PRINT & END) on one line.]

```
run
        FRIDAY, FEBRUARY 7, 1975 12:11:40PM-PST
        10 REM X IS: THE USER S NUMBER
        20 PRINT "TYPE A NUMBER."
        30 INPUT X
        40 PRINT "HERE IS YOUR NUMBER"
        50 PRINT X
        99 END
        *
        output: TYPE A NUMBER.
        input: 5
        output: HERE IS YOUR NUMBER
        output: 5
        execution completed at line 99
```

[He has now entered a program and uses the RUN command to try it out.]

```
more
        FRIDAY, FEBRUARY 7, 1975 12:12:07PM-PST
        17
```

[He was satisfied, so he says MORE to have BIP check the program.]

```
verify
        input: 1776
        output: 1776
        program run successfully
```

[The solution checker executes the program with some test data and decides it is ok.]

```
pti
        understand?: no
        skills: 55*  2*  13*  3
```

[The post-task interview: he says he did not understand the model solution and asks for more work on skills 55, 2, and 13.]

Figure 6 (cont.)

```
task
        FRIDAY, FEBRUARY 7, 1975 12:14:19PM-PST
        2
        bip's choice
        technique:  1
        must:  2

                [The next task.  The request for work on skill 2
                 has been honored.  The technique level has dropped
                 to 1, the technique to which skill 2 belongs.]
scratch
        FRIDAY, FEBRUARY 7, 1975 12:15:57PM-PST

run
        FRIDAY, FEBRUARY 7, 1975 12:19:18PM-PST
        10 PRINT "STRING"
        99 END
        *
        output: STRING
        execution completed at line 99

mode
        FRIDAY, FEBRUARY 7, 1975 12:19:33PM-PST
        2

verify
        output: STRING
        SCHOOL  f
        stay in problem after verifier failure

                [The program is not accepted.  BIP was looking for
                 output of the word "SCHOOL", but what it got was
                 "STRING."  The student continues his attempt to
                 solve the problem.]

demo
        FRIDAY, FEBRUARY 7, 1975 12:21:19PM-PST
        2

run
        FRIDAY, FEBRUARY 7, 1975 12:22:13PM-PST
        10 PRINT "SCHOOL"
        99 END
        *
        output: SCHOOL
        execution completed at line 99
```

Figure 6 (cont.)

```
more
        FRIDAY, FEBRUARY 7, 1975 12:22:21PM-PST
        2

verify
        output: SCHOOL
        program run successfully

                [This time BIP liked the output of the student's
                 program.]

pti
        understand?: yes
        skills: 2

                [He understood the model solution and thought he
                 now had had enough work with skill 2.]
```

Figure 6 (cont.)

author is interested. Most frequently, only incomplete protocols are obtained, since the simulation is completed once some critical behavior is produced by the system. Interactive simulation is a primary source of data for developing dialog-centered generative CAI systems (e.g., Collins, Warnock, & Passafiume, 1975).

The essence of interactive simulation, like program testing, is operating the program with both typical and boundary-condition inputs that will cause program execution to follow the various branches of the conditional control structures it contains. As a trivial example of program testing, a program to rank two numbers must handle cases where the first number is smaller than, greater than, or equal to the second number. For complicated programs, such as intelligent CAI systems, it is not really possible to conceive all the alternative control structures or the inputs that would activate them. Nonetheless, interactive simulation beyond normal degrees of program testing is effective for detecting and correcting errors and inadequacies.

Interactive simulation involves "playing" at being a student. The author conceives of some of the types of responses he expects his system should make in reaction to particular types of student behavior, and he then simulates that student behavior as best he can. Let us illustrate the nature of this process with an example based on our further development of the BIP-I system.

In designing task-selection procedures, one of our concerns is that the "remedial" task selected following a student's failure to complete a task not only address the inferred cause of the failure, but also be sensitive to the student's prior level of successful performance. This capability can be evaluated with interactive

simulation. The method is to interact with the system for a series of tasks, always completing them with no errors; we then deliberately fail a task-- call it HARDTASK. The task selected by BIP-I after HARDTASK is then examined carefully: Does it require the new skills introduced by HARDTASK? (We are assuming that the "failure" is due to those skills that were new to the simulated student when HARDTASK was selected.) Are the other skills it requires as advanced as those required by the task selected before HARDTASK? Next, we simulate a second student who begins the course and, unlike the first student, has consistent difficulty by failing tasks and requesting further help. We continue simulating this pattern of behavior, until BIP-I eventually selects HARDTASK. When it does, we fail HARDTASK in the same way as we did in simulating the first student, and then examine the next task selected. In the extreme case that the task selected after HARDTASK is the same in both instances, there is obvious inadequacy in the selection process. Otherwise, we must use the available evidence to judge whether the attempted remediation was sensitive to the differences between the simulated students. If we decide the choices were pedagogically adequate, then we probably want to repeat the simulation process a few times, varying the particular task we have the simulated "bright" student fail. In this manner, either we will become satisfied that the task-selection procedure provides sensitive remediation and move on to evaluate another feature of its behavior, or we will find a case where we judge the remediation to be inadequate.

When we judge that the system has behaved inadequately, our next goal is to determine the source of the shortcoming. To do so, we proceed to examine the states of BIP's data structures prior to the

problem and trace the execution of the task-selection algorithm in that context: Which skills were found to need work? Which tasks were considered? Which criteria determined the final choice? One approach we have used for finding appropriate modifications in these situations is to determine from a subjective examination of the available tasks themselves the task or tasks that we think would be reasonable to select. We then look for plausible changes to the procedure that would result in its choosing one of these tasks in that situation. Once any modification is made, it must be evaluated, with special attention to bad side-effects; for instance: does a change intended to improve remediation adversely affect the overall rate of progress for a student who never fails a task?

Interactive simulation has played a major role in the development of the BIP-II system (see Section V). One feature of BIP-II is the generation of inferences by which a given skill might be deemed "too easy" to be sought explicitly. Interactive simulation during the development of BIP-II had demonstrated clearly that a student enjoying consistent success would, in our judgment, progress through the curriculum too slowly. The task-selection procedures were considering all "as-yet unseen" skills to be candidates for the "needs work" set and looked for those skills, one by one, in the tasks to be presented. In many cases, a skill that we as instructors would consider to be too easy (given the student's progress) was being presented in an isolated context (e.g., in a task requiring very few other skills) because that skill was considered to need work. Since the appearance of such a "too easy" skill in a more demanding context would be more appropriate, we made this change: to allow tasks with "too easy" skills to be presented,

but not to allow such skills to be sought explicitly. That is, skills inferred to be too easy, given the student's past success on related skills, would not be put into the "needs work" set. It should be emphasized that the possibly unmotivating sequence of tasks originally produced for a successful student did not involve any implementation errors. Rather, the pedagogical adequacy of the original design was questioned, and then improved and re-evaluated by means of the interactive simulation.

Techniques for facilitating interactive simulation

Interactive simulation is a tedious process because of the time required to emulate complex patterns of actual student behavior by hand and to analyze the computations underlying a specific system response. Both aspects of the task can be made easier through the use of modified minimal instructional systems, interactive software debugging aids, and articulate user interfaces.

Modified minimal instructional systems. Often, many parts of a CAI system, including the normal "front end" which communicates with students, are not essential for producing the system behavior to be analyzed in the course of interactive simulation. Removing or modifying the nonessential elements can therefore facilitate the evaluation process. For example, in exercising a problem-solving laboratory that is intended to provide the student with critiques of his reasoning strategies, that capability can be examined sufficiently with a system modified to accept as input coded descriptions of reasoning behavior; the modified system saves the author the time required to produce actual problem solutions and eliminates the cost of executing all the

procedures that are required to infer reasoning strategies from actual problem-solving behavior.

In the case of studying task selection by BIP, we need not write the solution programs for the tasks, nor answer the self-evaluation questions asked in the Post Task Interview, since the procedure that updates the student model requires only a summary of the student's performance and his answers to the questions. Therefore, when we have engaged in interactive simulation, we have used a modified minimal instructional system that contains only the data structures and procedures essential for task selection and a special user interface that accepts coded descriptions of the simulated student's behavior. Thus the interaction consists solely of a number typed by the system identifying the task it has selected, and our response indicating the degree of difficulty the simulated student experienced in completing the task and the skills for which he specifically requested further work. Information about the tasks selected by the system (e.g., the skills it requires and the student's present state of learning) that is needed for analyzing specific aspects of the selection procedure's behavior could also be output regularly by the special user interface, or could be obtained selectively using the methods to be described in the next subsections.

Interactive software debugging aids. The analysis of a system's computations in a particular context can be simplified by the use of the interactive debugging facilities available in the powerful computer software systems in which most prototype, knowledge-based CAI systems are implemented. The most useful mechanisms include dynamic insertion of "breakpoints," which enable the author to control computation by

specifying that it is to be suspended whenever specific procedures (functions) are called by the CAI system. During the "break," system data structures, including the procedure itself and the current values of its parameters, can be examined and altered interactively before the author lets the computation resume.

For example, an interactive simulation with a modified minimal BIP-II system[3] might focus on task-selection behavior in situations where no troublesome skills are found in the student model; that is, at points when the system will be attempting to introduce the student to skills he has not used before. In order to avoid examining every call to the task-selection procedure in order to identify and further analyze the cases of interest, a breakpoint can be inserted to interrupt the selection procedure only when the subprocedure that assembles the set of "troublesome" skills returns an empty set. After a break occurs, the remainder of the task-selection procedure can be executed one step at a time to determine which structures in BIP-II's curriculum network are searched in assembling a set of skills that the simulated student is ready to learn, and which tasks involving those "ready" skills are considered and rejected before a task is selected.

In the course of this step-by-step execution, we might note that a particular skill is marked "ready" because its prerequisite skills (see Section V) have been learned, but judge that presentation of a task involving that skill would be premature in light of the simulated student's overall progress through the curriculum. Therefore, we might decide to examine more closely the basis for having defined the

---

[3] BIP is implemented in the SAIL dialect (Reiser, 1976) of ALGOL 60. We have used the BAIL debugging facility (Reiser, 1975) for interactive simulations.

prerequisite relations that affected the selection of that skill. Before modifying those relations, we would want to insert breakpoints that would take effect each time they affected a task-selection decision, so that we could judge their adequacy in a number of other contexts. In this manner, existing debugging systems can enable the author of a CAI system to analyze interactively only those interactions that are relevant to his examination of some specific aspect of his system's pedagogical behavior.

Articulate user interfaces. An articulate user interface maintains a representation of the instructional system's prior internal states sufficiently detailed for it to respond to queries about the reasons for its behavior. Optionally, the interface may also enable the author to modify interactively the instructional program with a high-level command language. Essentially, an articulate user interface is a customized interactive debugging facility, as described above. It enables the author to find out why the system behaved in particular ways without inserting breakpoints explicitly in procedures or interpreting data structures coded for machine, rather than human, processing. It also enables him to make modifications dynamically by specifying changes in a conceptual language that the articulate module executes on the actual data structures and procedures that comprise the instructional system. The first intelligent CAI systems with articulate capabilities that might be useful for interactive simulation have recently been developed (Brown et al., 1975), but there exist more advanced examples in other areas of generative programming such as question-answering systems (e.g., the RITA system by Anderson and Gillogly, 1976). Figure 7 is a hypothetical interaction with a non-existent version of BIP that is articulate about its task-selection procedure.

...

BIP: I choose Task 17.

Author: What are the skills in Task 17?

B: Skills are (55 2 13 3).

A: Which of those skills were you searching for in selecting a task?

B: (55 13)

A: Were there other skills you wanted to present?

B: (14 15 16 30 74)

A: Isn't there a task involving both Skills 13 and 14?

B: Task 47, "Wordy", involves Skills (55 2 13 80 62 14 52 4).

A: Why didn't you choose Task 47?

B: Skill 62 in Task 47 has as a PREREQUISITE Skill 42, which is in learning state N1 (unseen).

A: Is there a task involving Skill 14 that could have been chosen?

B: Task 18 is one such task.

A: Then why did you choose Task 17 instead of Task 18?

B: Both tasks had 2 skills I was looking for. However, Task 18 also involves Skill 4 which is already in learning state L2 (learned), whereas Task 17 involves Skill 3 which is in state N1 (unseen) and has no unsatisfied PREREQUISITE skills. Task 17 involved 2 troublesome skills and 1 new skill, but Task 18 involved only 2 troublesome skills. Selection Rule 3 tells me to choose the task with a maximum number of new skills that have satisfied PREREQUISITES when more than one task has the same number of troublesome skills.

...

Figure 7. Dialog between a course author and a hypothetical version of the BIP system that includes an articulate interface that enables it to describe its task-selection decisions.

The potential power of articulate systems is that they enable their authors to understand and manipulate the processes underlying the system's pedagogical behavior at a conceptual level, instead of in terms of data structures and procedures described in a programming language. The author always begins his implementation of an intelligent CAI system with a conceptual understanding of how its behavior will be produced. When he discovers his system's pedagogical inadequacies, he normally must understand and correct them first in terms of program structures and afterwards try to reformulate his conceptual models accordingly. By reducing the amount of thinking the author needs to do about his system's low-level programming constructs, the articulate user interface can help him focus on the evolving conceptual models realized by the operational CAI program.

Of course, the cost and special problems of incorporating an articulate system into an already complex CAI system are a formidable obstacle. For example, the articulate system must itself be computationally reliable before it can be safely used in evaluating the capabilities of the instructional system. The additional time and expense can be better justified if the articulate capabilities can be eventually integrated into the system when it becomes available to students. For example, we can imagine a friendly articulate version of BIP that allows the student to ask "Why" when it selects a task for him, and which is able to respond with "I thought you were having trouble with Boolean operators and this task gives an opportunity for more practice with them," or "This task introduces you to the use of FOR...NEXT loops which is a more advanced technique for iteration than the IF...THEN loops you have already learned."

Automated simulation techniques

In an automated simulation of a CAI system, the student or author-playing-as-student in the instructional interaction is replaced by a program that produces descriptions of student behavior. Each stage of the simulated interaction thus involves (1) the instructional program (I-Program) which generates some behavior, such as presenting text and related questions or problems, and (2) a simulation program (S-Program) which produces the answers, solutions, requests for assistance, or summaries of such responses (e.g., "incorrect solution: error-type 17") that will be used by the I-Program to determine its next behavior. For example, the goal in simulating the interactions of a generative CAI problem-solving laboratory might be to examine its ability to recognize and to correct when necessary the reasoning strategies used by students. In the simulation, each time the I-Program selected a problem, the S-Program would respond with a solution derived by a known strategy. The I-Program's subsequent analysis and commentary are the data by which its intended capabilities can be evaluated. The most obvious advantage of automated simulation over interactive simulation is the speed with which the I- and S-Programs can play out lengthy interactions and produce a corpus of simulated student protocols.

The uses of automatically simulated protocols in evaluation depend on the extent to which the overall patterns of behavior produced by the S-Program correspond to those of real students. It is certainly possible to have the S-Program vary its behavior by using an arbitrary decision rule that takes into account presumably important factors such as the difficulty of the questions and problems posed by the I-Program and the student behavior produced in prior interactions. The S-Program

in this case is a model of performance based on the author's intuitions about how students' behavior depends on their prior state of knowledge. These same intuitions are used by the author to guide his own behavior when he plays at being a student during interactive simulation. They also play a role in the I-Program itself, since they are embodied in the process that infers from the student's behavior what facts and skills in the student model are to be marked as "learned" and "not learned." Thus, an S-Program that is the author's intuitive model of performance cannot produce simulated data useful in evaluating the I-Program's rules for inferring changes to be made in the student model from student performance. Instead, the data can be used meaningfully only for checking computational correctness and pedagogical adequacy in the same limited, author-generated segments of interactions that can be examined by interactive simulation. Even so, automated simulation is useful because it can rapidly produce enough data for tabulations of the simple and conditional frequencies of the events recorded in the simulated protocols. These data may reveal previously unsuspected flaws in the instructional system's behavior. For instance, in evaluating a task-selection procedure, we can tabulate how often each task is selected after each other task as a function of the student behavior (e.g., success or failure) that occurs in response to the first task in the sequence. Odd patterns, such as one task always following another regardless of student behavior, or a very simple task frequently following a relatively complex task, signal possible program bugs or conceptual problems that can be investigated by tracing the task-selection procedure's computations for those segments in the simulated protocols.

An alternative approach to automated simulation is to use an S-Program that is a theoretical or empirical model of performance based on more than the intuitions of the author-- one that includes logically sufficient or statistical descriptions of how real students behave in a situation as a function of their state of knowledge. In this case, the simulated protocols record the I-Program's behavior in sequences of interactions where the simulated student's behavior is more likely to resemble that of a real student across a series of connected interactions. In particular, it becomes possible to examine interactions that might arise for real students in the later stages of instruction, which are difficult to anticipate and analyze during interactive simulation. With simulations using logically sufficient performance models, it also becomes possible to examine the adequacy of the I-Program's rules for inferring a student's knowledge from his behavior. Empirically based statistical models enable estimation of how student behavior will change with modifications made to the I-Program by generalizing the relationships between variables and student performance that have been observed during prior use of the CAI system.

Logically sufficient simulation programs. For some subject domains, it is possible to write "expert" programs that can synthesize answers and solutions for any of the questions and problems the I-Program presents to the student. The knowledge the student is to learn can therefore be represented in the I-Program's student model by the facts and skills embodied in the expert program (a procedural student model [Self, 1974]). The student's state of knowledge at any point of instruction is represented in the student model by marking as "learned" those facts and skills that the expert program must use to

produce the same answers and solutions the student has given up to that time. Carr and Goldstein (1977) refer to this as "overlay modeling," since the student model can be interpreted as an overlay on the model of an expert reflecting those facts and skills that the expert uses and that the student has yet to learn. Expert programs for most subjects are not easy to write. The problems involved are the subject of research in Artificial Intelligence. The use of overlay models in intelligent CAI has so far been in the context of some simple games, for example, "How the West was Won" (Brown et al., 1975) and "Wumpus" (Carr & Goldstein, 1977). These games have been embedded in instructional systems that use the output of expert programs to analyze weaknesses in the student's moves, and to tutor him on the simple computational and deductive reasoning skills required for expert play. Although these subject domains are simple, the systems that have been developed around them are among the best examples of how the ability to understand student responses, beyond merely judging their correctness, can enable sensitive tutoring in a CAI system.

For the purpose of simulation, student behavior can be produced with an S-Program that is a copy of the expert program with some of the facts and skills made inaccessible. The behavior of the S-Program can be interpreted as the behavior of a student who has yet to learn specific facts and skills of the subject he is studying. Simulation seems to have potential applications for assessing the I-Program's ability to infer from a student's behavior the overlay model that represents his state of knowledge, a capability the instructional system must have to individualize instruction appropriately. Although they do not elaborate their proposed methodology, Carr and Goldstein (1977)

mention plans to use simulation to evaluate generative CAI systems that model student learning with an overlay on an expert program. To make the procedure more comprehensible, we will outline one possible simulation technique that might be used for CAI systems with overlay models.

Most I-Programs incorporate a mechanism for representing the uncertainty present in inferring the student's underlying state of knowledge from his observed behavior. The uncertainty may reflect either suspected limitations in the I-Program's ability to analyze aspects of the student's behavior, alternative ways to answer a question or solve a problem that depends on different facts and skills, or assumptions about forgetting that "expect" a student to make some errors because he has temporarily lost access to facts and skills he has learned. Both the "How the West was Won" (Brown et al., 1975) and the WUSOR-II (Carr & Goldstein, 1977) systems represent the uncertainty of inferences about the student's knowledge of skills with a ratio of the number of times a skill is used in determining a move in the game to the the number of times the skill was required by better moves generated by an expert program. The ratio is compared to an arbitrary threshold in order to determine whether the skill is "learned" or "not learned." Thus, before a skill is marked as "not learned" (and thus in need of tutoring), the I-Program must observe that the student failed to use it in several situations where it should have been used.

One way to assess the adequacy of such arbitrary inferences about the student's state of knowledge is to ask him: If he complains frequently that the system is tutoring him about a skill he already knows, then the inferences need to be less conservative. Simulation

provides another approach, as follows. The S-Program is initialized to produce behavior that is a function of a specified overlay on the expert program, indicating some incomplete learning of the set of facts and skills used by the expert. It is thus able to answer some of the questions and solve some of the problems that the I-Program can present; the specific errors it makes are determined by the facts and skills it "does not know." The I-Program is initialized with its student model indicating that the student knows none of the facts and skills-- the only reasonable assumption given that it has no prior information about the student. The simulated interaction is begun and the I-Program analyzes the S-Program's answers and solutions, updating the student model and using it to determine its own behavior. Meanwhile, throughout the simulation the S-Program continues to produce behavior based only on the facts and skills it was given when it was initialized; that is, unlike most real students, the S-Program doesn't learn and improve its performance. The protocols produced by the simulation could allow the author to assess the I-Program's capabilities for analyzing behavior and inferring the knowledge it is based on by examining:

1) the situations in which the I-Program can successfully determine the overlay that enables the expert program to match the S-Program's behavior

2) whether, and if so how rapidly, the student model maintained by the I-Program <u>becomes</u> the same overlay on the expert that was initialized in the S-Program

3) whether the I-Program's own behavior provides interactions that teachers would judge reasonable for a student who behaved like the S-Program.

<u>Empirically-based simulation programs</u>. One problem with interactive simulation and automated simulation using expert programs is

that the simulated student behavior is unlikely to model the changes in the behavior of real students that are due to learning that occurs across a series of interactions with the CAI system. S-Programs derived from expert programs do not necessarily model real students because they are only logically sufficient models of how students perform given what they know, and are not logically sufficient models of how students learn what they know from the instructional system. Any changes introduced in the facts and skills accessible to the S-Program reflect only the author's intuitive model of how some students will learn by interacting with the I-Program. Consequently, the use of data from the simulation for evaluating pedagogical effectiveness of the I-program by measuring changes in the S-Program's ability to answer questions and solve problems is invalid. There is no way around this limitation unless an empirically derived model of student learning and performance is available: a model that describes the likelihood (1) that the student learns new facts and skills as a result of the I-Program's behavior, and (2) given that he does, his behavior is such that the I-Program can discern the new learning. The need for an empirical model implies of course that in order to use simulation to evaluate pedagogical effectiveness, the CAI system must have been already used by real students. In that case, one might ask what is the use of simulation, since protocols from real students provide whatever data are needed for evaluation?

Simulation can still be useful for evaluation even after real student data are already available. Typically, the data collected from the first use of a generative CAI system will reveal situations in which its behavior is inadequate and its effectiveness for stimulating new

learning is limited. The author will want to make modifications in his conceptual models and the CAI system itself which need to be evaluated. The possibility that we have considered in our research is that by using a statistical model of learning and performance derived from the existing real-student data, a simulation can produce protocols that will allow us to estimate the effects of modifications, while assessing their correctness and adequacy. Statistical models describe the probabilities for each student behavior that might occur in a given situation as a function of the parameters by which situations can be classified. For example, the situation surrounding the presentation of a problem might be characterized by the identity of the problem, the facts and skills required for its correct solution, and the student's state of knowledge for those facts and skills. The S-Program operates by recognizing situations created by the I-Program's behavior and then using the probabilities of different student behaviors in those situations obtained from the statistical model to constrain its selection of the student behavior to be simulated. Suppose, for instance, that the student were answering a question that requires knowledge of facts a and b, both of which are marked as "not learned" in the student model. The statistical model will be queried by the S-Program to obtain the probability p of a correct answer, as determined from data about the behavior of past students who were asked the same question when a and b were "not learned." The S-Program will then produce a correct answer with probability p.

The probabilities associated with every identifiable instructional situation are an empirical model of learning and performance for that situation. They describe the effects of the

student's state of knowledge as given in the I-Program's student model, modified (if at all) in the ongoing situation, and of any unknown mediating effects (forgetting, lack of attention, etc.) on his behavior. So, for example, the probability of a success on a problem that involves two skills that are "not learned" is really a representation of several complex joint probabilities, which cannot be determined separately by observation of the student's behavior. These probabilities are:

1) The probability that the inferences (that the two skills are "not learned") are correct.

2) The probability that if the skills are in fact "not learned," they will be learned in the course of working the problem.

3) The probability that, if the skills were learned either prior to or during the problem, the student will generate a correct solution (i.e., he understands the problem correctly, he does not forget any of the other skills the problem may involve, etc.).

Thus, a statistical model represents variability in performance that is due to learning and other unknown factors, and that can be introduced into logically sufficient models only via ad hoc mechanisms.

The issues to be confronted in designing simulations based on statistical models include defining the parameters that characterize instructional situations and assuming that a set of parameters adequate for one version of a CAI system will also be adequate for a modified version. The next section describes the details of our use of statistically-based simulation with the BIP-I system.

IV. Simulation with the BIP system

Rationale

We implemented an automated simulation system to be used in evaluating alternative task-selection procedures with the BIP-I system. Our goal was to develop a tool that would allow us to exercise modified procedures thoroughly enough to detect most errors and cases of pedagogically inadequate decisions, and to estimate measures of pedagogical effectiveness for alternative procedures. Use of the BIP system by real students had previously demonstrated the adequacy and effectiveness of its existing task-selection procedure for a range of different patterns of student performance (Barr et al., 1976). However, there were cases in which both we and students thought that BIP-I's decisions were inappropriate (see Section II). The problems could be traced to the criteria by which the student model was updated after completion of a task, and more generally to the lack of any detailed representation of the relationships between the many separate skills used to describe tasks and to model student learning. One specific problem involved the task selected when a student had quit the previous task without completing a solution. If the student had been advancing rapidly, succeeding on all the tasks prior to this one, then the next task selected occasionally involved skills he had previously used successfully, and did not emphasize the most difficult skills from the failed task. The problem was caused by the rules used to decrement the counters in BIP-I's student model reflecting success and failure with a skill, and by the criteria based on those counters for deciding that a

skill required more work. A few possible solutions were fairly obvious, but they needed testing to determine how they might affect situations for which the behavior of the task-selection procedure was already satisfactory. Interactive simulation was a necessary first step, but could not provide data about a sufficiently wide range of situations to detect unwanted side effects. Expense made it impractical to try out the possible modifications with groups of real students to determine the most effective change. We decided therefore to explore the use of automated simulation, using the available real-student data to build a statistical model that could interact with modified task-selection procedures and reveal their behavior for a range of situations. We also planned to use the simulation in developing the BIP-II system, embodying a major revision of the CIN representation of BASIC programming knowledge (See Section V). We knew that that revision would include the use of data structures and algorithms considerably more complex than those used in the BIP-I task-selection procedure and would therefore involve extensive testing and modification. At the same time, this use of the simulation would test the extent to which data collected from real students under one version of an instructional system can be used to estimate data that would result from the use of related systems.

Overview

BIP-I's task-selection procedure uses information from the CIN, which represents the programming skills and the tasks, and the student model, which indicates the learning of each skill inferred to have occurred from work on previous tasks. In updating the student model, BIP-I uses information from the solution checker, which is called by the

student when he feels his program meets the requirements of the task, and from the post-task interview, in which the student is asked whether he understands the "model solution" for the task and whether he wants more work on each of the skills involved. A simulation program for interacting with the task-selection procedure thus needs only to produce a description of the outcome of a task, indicating whether the simulated student completed a correct solution or quit the task, and the answers he gave to the yes/no questions asked in the PTI.

Task selection is based on criteria by which troublesome skills, or skills that a student is ready to learn, are identified. The implicit assumption is that the effectiveness of presenting a task for stimulating new learning depends only on the prior learning of the skills in that task. While clearly an oversimplification of the relationships between prior learning and the new learning and performance that result from working a task, BIP-I's criteria reflecting this assumption have produced adequate task selection in most cases. Our simulation program therefore accepts as input from the task-selection procedure a description, which we call a configuration, of the prior learning of each skill in the task that has been selected. The simulation program generates its output by using a statistical model of the relationships between configurations and task outcomes determined from data on previous use of the BIP-I system by real students. These data are condensed protocols consisting of sequences of task identifiers and coded descriptions of the task outcomes for each student. The output of the simulation program is a condensed protocol, identical in format to those used to build the statistical model, indicating the sequence of tasks chosen by the selection procedure and the outcomes

generated by the simulation. Each protocol represents one simulated student who works on the tasks selected by BIP-I until the student model indicates that all the skills are learned, or that there are no more available tasks involving the remaining unlearned skills.

Finite-state student model

In implementing the BIP-I simulation, an improved learning model was included. In the BIP-I system operational before that time, the student model consists of a set of counters associated with each skill, indicating how many times the student had used the skill, how many times he had been successful in the tasks that required it, how many times he had responded with confidence in his own ability to the post-task interview, etc. These counters are used to determine whether or not the student needs more work with the skill at the time a next task is to be selected. While counters seem simplistic, they do work and are still used in some of the latest AI-based generative CAI systems.

A more sophisticated approach is to describe the student's knowledge of each skill with respect to a set of states. The names of the states may have either psychological significance ("learned", "not learned") or pedagogical significance ("ready to be learned", "too easy for the current context"). The finite-state model has both conceptual and computational advantages. Pedagogical heuristics can be conceived in terms of meaningful categories of learning, instead of counter values. Transitions between states are simpler to implement and modify than are non-unitary increments and decrements of multiple counters. The limited number of states can make it easier to implement more complex algorithms for the task-selection process. In addition, a

possible extension (which we have not attempted) is that state transitions can be made probabilistic, enabling the application of existing "technology" of finite-state Markov learning models from mathematical psychology.

We defined a non-probabilistic six-state model to be incorporated into experimental revisions of the BIP-I task-selection process. Initially, for the purposes of simulation, the model was designed to be functionally equivalent to the existing counter model--i.e., the transitions between states parallel the changes made to counter values for given student behaviors in completing tasks. The six states and their meanings are:

N0 -- Skill has not been presented, nor have others at its technique level.
N1 -- Skill has not been presented, but others at its level have been seen.
U0 -- Skill has been presented but has not been learned.
L3 -- Lowest level of learning. Skill was required in a task in which the student had difficulty achieving an acceptable solution.
L2 -- Skill is considered "learned," having been used successfully but in a restricted context of other skills.
L1 -- Highest learned state. Skill has been used successfully in varied skill contexts.

Simulation database design

The statistical model used by the simulation program is a database of discrete entries, one for each distinct configuration of prior skill learning identified in the real student protocols. Each entry contains the empirical probabilities for the possible task outcomes observed for all tasks described by its configuration. The format of a configuration is a list[4] in which each element consists of a skill learning state (e.g., "new", "unlearned", "well-learned") and the

---

[4]LISP conventions will be used for denoting list structures.

number of skills in the task that were in that learning state when the task was presented. We represent a configuration as

$$((S_1 \cdot n_1) (S_2 \cdot n_2) \ldots (S_m \cdot n_m))$$

where the S_i are different learning states, the n_i are the integer counts of the number of skills in a state, and m is the number of different states that were associated with at least one skill in the tasks described by the configuration. For instance, all cases in the real student protocols that involved three skills, two of which were in learning state N1 (new) and the third in state L2 (learned), are represented in the database by an entry for the configuration

((N1 . 2) (L2 . 1))

The probabilities for each outcome observed to have occurred for a skill configuration across all protocols are given in the database by a list of the frequencies of each outcome. Continuing the previous example, the data stored with the configuration ((N1 . 2) (L2 . 1)) might be

```
( ((SUCC . 10) (DIFF . 8) (QUIT . 2))
  ((YES 17) (N1 24 . 40) (L2 0 . 20)) ) .
```

The first line denotes the EVENT, a summary of the student's performance in completing a solution: SUCC (success without difficulty) occurred in 10 tasks, DIFF (success with difficulty) in 8 tasks, and QUIT (failure to complete a solution at all) in 2 tasks. The sum (20) of the EVENT counts is the total number of times a task described by the configuration ((N1 . 2) (L2 . 1)) occurred in the student protocols. The second line of data gives the PTI responses. The first sublist, (YES 17), indicates 17 "yes" answers (out of 20) to the question "do you understand the model solution?" The remaining sublists are the

frequencies with which students answered that they wanted more work for the skills in each learning state. For the two skills in state N1, there were two questions asked in each of the twenty tasks represented by this database entry, and for those 40 questions there were 24 answers requesting more work on those skills. For the one skill in state L2, none of the 20 questions asked were answered with a request for more work.

In using the database during simulation, probabilities are computed dynamically by performing the appropriate divisions of observed by possible frequencies stored with the configuration that describes the task presented by the selection procedure. The reason for storing frequencies instead of probabilities is best explained by example. Suppose we thought that skills in state L2 (already learned skills) did not have effects of practical significance on outcomes of tasks involving those skills. So, for example, we expected the same distribution of outcomes for tasks described by configurations ((N1 . 2) (L2 . 1)), ((N1 . 2) (L2 . 5)), ((N1 . 2) (L2 . 15)), etc. By storing frequencies in the database, the simulation can be used to test this hypothesis about the irrelevance of skills in state L2. We can compare the results of simulation experiments where L2 skills are and are not included in determining the configuration that describes each task. In the latter case, the probabilities used by the simulation program are computed by pooling the frequencies across all entries with identical counts on all learning states except state L2. Thus, if in ignoring the L2 state, a configuration of ((N1 . 2)) is determined for a task, then all the entries listed above, corresponding to configurations of two skills in state N1 and of any number in state L2, will have their

frequencies added together. If the comparison of simulated data from experiments where states were and were not ignored reveals no differences of practical significance on variables such as average number of tasks worked by each student, number of tasks failed, number of requests for more work on skills, etc., then we can conclude that the skills in the learning states that were ignored do not affect the performance we intend to improve by individualizing task selection. The implication is that the task-selection procedure need not base its decisions on skills in those states that can be ignored during simulation. Thus, in addition to its originally planned use in evaluating modifications to task-selection procedures, our automated simulation program can be used to analyze data from the use of existing procedures to suggest possible modifications to them.

Protocol analysis procedure

The complete protocols used in constructing the database are a chronology for each use of a BIP command by students (see Figure 6).[5] These commands include requests for hints, requests to review task descriptions, calls to debugging aids, etc., that do not affect the task-selection procedure and do not necessarily indicate whether or not the student is having difficulty. For task selection, the relevant commands are TASK, which requests a new task, and MORE, which calls VERIFY (the solution checker) and PTI. The MODEL command, which indicates a request to examine the model solution before the task is completed probably does reflect student difficulty, and is of interest for characterizing performance even though the task-selection procedure

[5] More recent protocols also include each line of BASIC typed by students in writing and debugging their programs.

in force when our real student protocols were collected did not monitor it.

The first stage of the protocol analysis used to create the simulation database was accomplished by a scanning program, STRAIN, that produces condensed protocols containing the sequence of interactions for selected BIP commands. Using STRAIN, we obtained protocols listing only TASK, MODEL, and MORE, including its calls to VERIFY and PTI (Figure 8).

The second stage of protocol analysis involved a second scanning program, FRAMER, which scans the condensed protocols written by STRAIN. FRAMER finds the beginning of each task and first determines the EVENT part (SUCC, DIFF, or QUIT) of the task outcome. Although the version of BIP that was used by our real students distinguished only success (SUCC) and failure (QUIT) in updating the student model, we defined a third category of success-with-difficulty (DIFF), which should indicate instances of challenging, but manageable tasks. In FRAMER, DIFF is determined by a request to examine the model solution or by one or more rejections of the student's program by the solution checker prior to its being accepted as correct.[6] SUCC represents writing a program accepted by the first call to the solution checker and QUIT represents leaving the task without ever having "passed" the solution checker. After determining the EVENT, FRAMER finds the PTI responses. It first scans the yes/no answer to the understand-the-model-solution question, and

---

[6] In more recent use of the simulation, not described in this report, two or more rejections by the solution checker are used in defining DIFF. Hand analysis of student protocols in the context of a research program on debugging has indicated that students frequently initially write programs that reflect misunderstanding of the task specifications--they write a correct program to solve the wrong problem. Since this does not correspond to any difficulty with programming knowledge per se, we felt a relaxation of the criterion for defining DIFF was in order.

```
task
        FRIDAY, FEBRUARY 7, 1975 11:59:10AM-PST
        17
        bip's choice
        technique:  5
        must:  13  14  15  16  30  55  74

model
        FRIDAY, FEBRUARY 7, 1975 12:06:53PM-PST
        17
        10 INPUT X
        *

more
        FRIDAY, FEBRUARY 7, 1975 12:12:07PM-PST
        17

verify
        input: 1776
        output: 1776
        program run successfully

pti
        understand?: no
        skills: 55*  2*  13*  3

task
        FRIDAY, FEBRUARY 7, 1975 12:14:19PM-PST
        2
        bip's choice
        technique:  1
        must:  2

more
        FRIDAY, FEBRUARY 7, 1975 12:19:33PM-PST
        2

verify
        output: STRING
        SCHOOL  f
        stay in problem after verifier failure

more
        FRIDAY, FEBRUARY 7, 1975 12:22:21PM-PST
        2

verify
        output: SCHOOL
        program run successfully

pti
        understand?: yes
        skills: 2
```

Figure 8. Condensed BIP protocol produced by running the STRAIN program over the protocol section given in Figure 6.

then the list of skills for which further work was requested. The output of FRAMER for each condensed protocol is a sequence of frames, which are coded descriptions that succinctly identify each task and its outcome (Figure 9). For example, the frame (TK017 DIFF (NO SK055 SK002 SK013)) records that on Task 17, the EVENT was success-with-difficulty, and in the PTI the student said he did not understand the model solution and wanted more work on Skills 55, 2, and 13.

The simulation database was constructed from the sequences of frames by a third program, ANALYSIS. ANALYSIS includes a copy of BIP-I's CIN and the procedure used to change the student model, modified to accept protocol frames as its input, and operates as follows:

1) The student model is initialized and the frames for one student protocol are retrieved.

2) For each frame, the task number is used to enter the CIN and retrieve the skills involved in that task.

3) A list of the learning states of those skills is determined from the student model.

4) The configuration for the list of states is computed.

5) The database entries defined to that point are searched for that configuration. If an entry already exists, the EVENT and PTI from the frame are used to increment the appropriate frequencies stored there; otherwise, a new entry corresponding to that configuration is added to the database.

6) The outcome of the frame is used to update the student model before scanning the next frame.

After the frames for one student are completed, the student model is reinitialized before retrieving the frames for the next student. ANALYSIS outputs the database when all the student protocols have been scanned.

```
 (TK001 DIFF (YES))
 (TK072 QUIT (YES))
 (TK072 SUCC (YES))
 (TK003 SUCC (NO SK005))
 (TK004 SUCC (YES))
 (TK005 SUCC (YES))
 (TK006 DIFF (YES))
 (TK009 SUCC (YES))
 (TK014 SUCC (YES))
 (TK015 DIFF (YES))
 (TK016 DIFF (YES))
 (TK007 SUCC (YES))
 (TK085 DIFF (YES SK082))
 (TK076 SUCC (YES))
 (TK077 SUCC (YES))
 (TK079 DIFF (YES SK030 SK074))
*(TK017 DIFF (NO SK055 SK002 SK013))
*(TK002 SUCC (YES))
 (TK020 DIFF (YES SK055 SK002 SK013))
 (TK010 SUCC (YES))
 (TK011 SUCC (YES))
 (TK090 SUCC (YES))
 (TK089 SUCC (YES))
 (TK028 SUCC (YES SK055 SK002 SK029))
 (TK021 SUCC (YES SK055 SK002))
 (TK018 SUCC (YES))
 (TK019 SUCC (YES))
 (TK022 SUCC (YES))
 (TK023 SUCC (YES))
 (TK083 SUCC (YES SK027 SK019))
 (TK080 SUCC (YES SK029))
 (TK031 SUCC (NO SK036))
 (TK033 SUCC (YES SK039 SK038 SK076 SK075))
 (TK037 QUIT (YES SK055 SK013 SK042 SK029 SK059))
 (TK081 DIFF (YES SK024 SK030))
 (TK082 DIFF (YES SK023 SK030))
 (TK030 SUCC (YES SK018))
 (TK024 SUCC (YES SK016))
 (TK061 DIFF (YES SK013 SK046 SK036 SK059))
 (TK035 DIFF (NO SK042))
```

Figure 9. Task frames produced by the FRAMER program. The starred lines are the frames for the two tasks given in Figure 8.

The student protocols used to construct the database used in the experiments to be described here were those available from nineteen students who used BIP-I with its task-selection procedure in the context of a previous experiment (Barr et al., 1976).[7] The nineteen protocols were described by 678 task frames, an average of 36 tasks per student. The protocols were incomplete in that, for the purposes of the experiment, each student had been limited to ten hours on the system, and thus some students did not reach a point where BIP-I decided it had no further tasks to present to them. From the protocols, ANALYSIS created a database of about 200 entries. The number of entries relative to the total number of tasks emphasizes the number of different situations in which BIP-I's task-selection procedure must perform adequately. However, it also implies that many entries contain data from just a few, or even just one, task(s) worked by a few (one) students. The spread of the data across so many different configurations was one factor that motivated us to pool entries during simulation as described above.

The error inherent in the statistical model represented by the database decreases as the number of real student protocols adds to the amount of data represented by each entry. Given enough data, we would want to add task identity as an additional parameter of the configurations used to define entries. Each entry would then correspond to a particular task, and hence its particular skills, and there would be several entries for each task corresponding to the different configurations of learning states in the different cases when it was

---

[7] These students were Stanford undergraduates with non-technical backgrounds and no previous programming experience.

selected. The advantage would be that any differences between specific skills and the tasks themselves (e.g., the semantic content of the programming problem) that also affect student performance would be accurately reflected in the simulated outcomes.

Simulation procedure

The SIMULATION program interacts with a task-selection procedure to produce sequences of task frames with the same format as those read by the ANALYSIS program to create the database. The sequence of actions for simulating one student is as follows:

1) The student model maintained by the task-selection procedure is initialized.

2) The selection procedure selects a task based on the learning states of skills in the student model.

3) The configuration of learning state counts describing that task is computed.

4) The entry in the database corresponding to that configuration, or, if none exists, to the closest matching configuration, is retrieved.

5) The frequencies for the possible EVENTS are converted to probabilities and a random number is mapped onto the probability space to determine the EVENT to be simulated.

6) The frequencies for each PTI question are converted to probabilities. For each question, a separate random number is generated and mapped onto the appropriate probability space to determine an answer.

7) The simulated outcome is read by the task-selection procedure to update the student model, and the simulated frame is recorded, before selecting a next task and repeating the simulation process.

With regard to step 4, it sometimes happens that there is no entry in the database corresponding to the exact configuration of the task to be simulated. In this case, the data to be applied are taken

from the "closest" matching entry. The algorithm used to find a match operates by successively decrementing by one each count in the task configuration, checking each time for a matching entry in the database. If none is found, it decrements by one the possible pairs of counts, then triplets, and so on. If no match has been found, it then decrements by two each count and calls itself recursively with a decrement of one on the other counts. The decrement size is increased until a match is found. The algorithm is complex for reasons of completeness; in practice, a match is usually found after a few small decrements to the original target configuration.

To pool entries in a simulation experiment, SIMULATION is told to ignore specified learning states in computing configurations. Then when database entries for a configuration are retrieved, all entries with the target configuration and any other counts of the ignored learning states are pooled by adding their frequencies.

As a test of the simulation procedure, it was first used with the task-selection algorithm under which the real-student protocols had been collected. The differences between the original and simulated sequences of task frames are a measure of the error in the statistical model of learning and performance represented by the database. There is an expected error component due to differences between specific skills and tasks that are ignored when predicting outcomes solely from the inferred learning states of the skills involved in tasks. Another source of error is individual differences between students that are lost by combining their data in a single database; these differences are such that we do not feel it accurate to assume they represent normal variability from a single statistical population. For example, some

real students seldom, if ever, ask for more work on skills during the PTI, while others ask for more work continually, regardless of the difficulties they may be having in completing tasks. For these students, PTI responses do not seem to reflect any real problems with skills, but instead to indicate different learning styles and strategies. Since the simulation assumes that variability among students is normal, it will not produce output corresponding to extreme variations in student performance as frequently as they occur in the population of real students. These known sources of error in prediction prevent SIMULATION from generating data identical to that of real students when used with an identical task-selection procedure. However, the observed discrepancies can be used to gauge the minimum error of prediction expected from simulation with modified selection procedures. Thus, for example, if there is a reduction in the average number of tasks failed under a modified procedure that is greater than the difference for that measure observed between real and simulated use of the original procedure, then we might conclude that the modification was effective in reducing the measure.

In the course of using SIMULATION with the original task-selection procedure, the "closest match" algorithm was developed and several assumptions for pooling entries were tested. We then moved on to simulate two modifications of the original selection procedure. One involved the way in which the skills in the student model are updated after a failure to complete a task. The other involved the task-selection criteria regarding the number of troublesome or new skills to be included in the next task.

Our general procedure in using SIMULATION was to run two

identical simulated experiments of twenty students each for each modification of the task-selection procedure or of SIMULATION itself. Thus, in comparing the results of different simulation experiments with each other and with the real student data, differences could be evaluated relative to the error in the database, as determined by measuring the differences between two identical simulation experiments.

Results and discussion

Data for the nineteen real students and each simulation experiment are presented in Tables 1, 2 and 3. Table 1 gives

1) the mean number of task frames per student

2) the proportion of instances for each type of EVENT (SUCC, DIFF, QUIT)

3) the proportion of instances for PTI responses indicating understanding of the model solution and requests for further work on skills

4) The proportion of instances in which a "break" occurred in selecting a task.

A break is defined as the selection of a task more than or less than one technique level away from that of the previous task, and is a measure of the continuity of the sequences of tasks that are selected for a student. The greater the proportion of breaks, the more the student "bounced" back and forth between tasks involving predominantly simple and complex skills.

Table 2 gives the conditional probabilities for successive task EVENTs: for example, the probability that when task n-1 was a SUCC, task n was a QUIT. These probabilities reflect the ability of the selection procedure to maintain a challenging level of difficulty.

Table 3 gives for each pair of experiments the value of a

Table 1

Task Outcomes from Real and Simulated BIP Protocols

	EXPERIMENTS								
	Real Students	1a	1b	2a	2b	3a[2]	3b[2]	3c[2]	3d[2]
Mean No. of Tasks per Student	35.7[1]	48.4	47.2	46.0	46.8	48.0	50.7	49.5	51.1
Proportion:									
SUCC	.60	.57	.59	.58	.57	.56	.54	.56	.54
DIFF	.87	.41	.39	.41	.41	.42	.43	.41	.43
QUIT	.03	.02	.02	.01	.02	.02	.03	.03	.03
Understood model	.96	.97	.94	.96	.97	.96	.93	.95	.97
Skill requests	.11	.08	.08	.08	.09	.12	.13	.13	.14
Breaks	.18	.24	.20	.22	.21	.20	.21	.18	.18

[1]Students were limited to 10 hours of system time. Many of them did not reach a point where the task selection procedure decided they had finished the curriculum.

[2]Proportions based on only the first 36 tasks of each simulated sequence.

Table 2

Conditional Probabilities of Task EVENTS
from Real and Simulated BIP Protocols

Task Event		EXPERIMENTS								
task n-1	task n	Real Students	1a	1b	2a	2b	3a	3b	3c	3d
SUCC	SUCC	.64	.59	.60	.59	.58	.56	.54	.57	.56
SUCC	DIFF	.33	.39	.39	.39	.40	.41	.43	.40	.41
SUCC	QUIT	.03	.02	.01	.02	.02	.03	.03	.03	.03
DIFF	SUCC	.54	.53	.56	.59	.58	.54	.54	.54	.49
DIFF	DIFF	.42	.45	.42	.41	.39	.45	.42	.44	.49
DIFF	QUIT	.04	.02	.02	.00	.03	.01	.03	.02	.02
QUIT	SUCC	.79	.92	1.00	.62	.44	.87	.77	.83	.94
QUIT	DIFF	.21	.08	.00	.38	.50	.13	.23	.17	.06
QUIT	QUIT	.00	.00	.00	.00	.06	.00	.00	.00	.00

77

Table 3

Values of R_1 and R_2 for Task Sequences
from Real and Simulated BIP Protocols

	Real Students	1a	1b	2a	2b	3a	3b	3c	3d
Real Students	---	.92 .82	.92 .79	.90 .78	.90 .81	.91 .73	.94 .74	.95 .82	.94 .80
1a		---	.99 .92	.99 .92	.98 .93	.97 .83	.96 .82	.98 .89	.98 .89
1b			---	.98 .91	.98 .92	.96 .82	.95 .80	.97 .89	.97 .89
2a				---	.98 .91	.96 .84	.94 .77	.97 .89	.96 .89
2b					---	.96 .83	.94 .80	.96 .89	.96 .86
3a						---	.97 .89	.95 .80	.96 .87
3b							---	.96 .80	.97 .85
3c								---	.99 .91
3d									---

statistic we denote $\underline{R}_n$ for n = 1,2. For subsequences of tasks of length n, $\underline{R}_n$ is an index of correlation that indicates the extent to which the subsequences that occur in one experiment occur with the same frequency in a second experiment. The upper bound of $\underline{R}_n$ is 1.0, the case in which every subsequence occurred in both experiments with equal frequencies; the lower bound is 0.0, the case in which no subsequence occurring in one experiment also occurred in the other. When n = 1, $\underline{R}_1$ measures the relative frequency with which each task in the curriculum was presented in the two experiments. The formal definition of $\underline{R}_n$ is given in Appendix B.

Experiment 1. Two identical experiments, 1a and 1b, were simulated using the task-selection procedure under which our real student data had been collected.[8] We actually ran many experiments with the original procedure in order to determine whether there was a basis for pooling entries in the database. Experiments 1a and 1b are cases in which we pooled entries by ignoring states L1 (well-learned) and L2 (learned) in computing learning state configurations for each task. These states represent the greatest degrees of learning for skills; thus by ignoring them, the simulation program assumed that outcomes are not significantly influenced by the number of already-learned skills involved in a task.

The data given in Table 1 for Experiments 1a and 1b can be compared to gauge the error present in the simulation procedure. The

---

[8]One change to the procedure was the substitution of the six-state learning model for the counter variables originally used in the student model to represent the learning of each skill; the change was transparent since the transitions for the six-state process were designed to be equivalent to the increments and decrements of the counter variables.

maximum variability between simulation runs is .04 for breaks and the minimum is .0 for QUIT and skill requests. In comparing the proportions obtained by simulation to those of the real students, the differences seem to fall within the variability observed between the two simulation experiments. The conditional probabilities for Experiments 1a and 1b in Table 2 display greater variability than the data in Table 1. This is to be expected since each conditional probability is a finer breakdown of the EVENT data based on fewer instances. In comparing the conditional probabilities of the simulated and real-student data, the largest differences occur when task n-1 is a QUIT; the simulation overestimates the probability of a following SUCC.

The values of $\underline{R}_1$ and $\underline{R}_2$ given in Table 3 for comparison of the two simulated sets of task sequences again allow us to gauge the variability inherent in the simulation. For 1b vs. 1a, $\underline{R}_1 = .99$ and $\underline{R}_2 = .92$; We would not expect the values obtained in comparison with the real-student data to exceed these values obtained for repeated simulations. Table 3 indicates that the values of $\underline{R}_1$ and $\underline{R}_2$ are lower for either simulation vs. real-student comparison. Thus, the simulated sequences do seem to deviate from the sequences obtained from the real students. Since $\underline{R}_n$ is a weighted average of components for each identifiable subsequence (see Appendix B), we were able to examine the breakdowns of $\underline{R}_1$ and $\underline{R}_2$ to determine the source of the difference between the real and simulated data. We found that the overall differences could be attributed to a few tasks that occurred once each across all the real-student data and never occurred at all during either simulation experiment. Tasks and subsequences of two tasks that occurred frequently in the real-student data occurred with comparable

frequency in the simulated data and did not contribute systematically to the differences observed in the overall values of $\underline{R}_1$ and $\underline{R}_2$. The appearance of a task only once in the real-student data probably reflects extreme variability in student behavior that the simulation cannot reproduce with the same frequency as occurs in the real-student population. For example, if one real student made it a habit to request more work on skills regardless of his success in completing tasks, then in honoring those requests the selection procedure might choose some tasks that were never selected for any other student. Our analysis of the breakdown of $\underline{R}_1$ and $\underline{R}_2$ by tasks suggests that one or two cases of atypical student behavior could account for the observed differences between the values of $\underline{R}_n$ for real-students and Experiment 1.

In general, the comparison of data from real students and the simulation in Experiment 1 indicates that with the same selection procedure, the simulation produces much the same results as were obtained with real students. Although the simulation used the number of skills in each learning state, and ignored both some of the states and the identity of specific tasks and skills, most of the differences between real and simulated measures fell within the variability observed in the simulation alone.

Experiment 2. In Experiment 2 (identical runs 2a and 2b), the task-selection procedure was modified with respect to how the learning states of the skills in a task are updated to reflect a QUIT. In the original procedure, when a QUIT occurred all the skills in the task were put into a state which caused those skills to be added to the MUST set when the procedure selected the next task; the MUST set contains those skills which the procedure attempts to have included in the next task.

The modification for Experiment 2 was to redefine the transitions between states so that skills in higher states of learning did not have their states changed after a QUIT, and so were not subsequently added to the MUST set. Thus, only the marginally learned and new skills in the failed task affected the next task selection.

Table 1 reveals no differences between Experiments 2a and 2b and either the real-students or Experiment 1. This is not surprising since the modification could only have an effect on the selection decisions following the 2-3 percent of tasks that were QUIT. The effect of the modification is seen in the entries in Table 2 for sequences where task n-1 was QUIT. For both Experiments 2a and 2b, there is a substantial decrease in the probability of a SUCC given a QUIT as compared to the real students and to Experiment 1. This is as expected, since previously well-learned skills are no longer being included in the MUST set after a QUIT and so the next task is more likely to contain skills that are more difficult for the student to use. The values of $\underline{R}_1$ and $\underline{R}_2$ in Table 3 show that the actual task subsequences produced in Experiment 2 are identical (within the variability of the simulation) to those of Experiment 1.

Experiment 2 demonstrates that the simulation procedure can make reasonable qualitative predictions for the effects of small modifications to a task-selection procedure.

Experiment 3. For simulation Experiment 3, an additional modification was introduced into the original BIP-I task-selection procedure. Let M be the number of skills in the MUST set which are included in a task chosen by the selection procedure. The original procedure tried to maximize M for every task-selection decision. Thus,

if there were six skills in the MUST set, the procedure would attempt to find a task in the curriculum that involved the use of those six skills. Letting M assume its maximum value insures that the procedure will always try to present the most difficult task consistent with the technique level at which the student has been working. The modification we tested in Experiment 3 was to make M a parameter of the selection procedure that is fixed at a specific value for a group of students. We were most interested in the case in which M is set to 1. Pedagogically, this corresponds to the principle of attacking the learning of troublesome and new skills by isolating them one at a time in the context of problems that involve only other already well-learned skills. We did not necessarily believe that presenting a task involving only one MUST skill is always better than presenting one with a maximum number of such skills. Instead, we thought that setting M equal to 1 might be useful for some students in some situations (e.g., after failing a task), but still were interested in determining the effects that would be predicted by the simulation for the extreme case where M always equals 1.

We conducted four separate simulation runs, 3a and 3b where M = 1, and 3c and 3d where M = 3. The proportions reported in Table 1 for these experiments reflect only the first 36 tasks in each simulated student sequence of frames. The sequences were truncated to provide a more appropriate comparison with the real-student data which, as we have noted, is based on an average of 36 tasks per student because of the limited system time allowed students in that experiment.

Surprisingly, the data in Table 1 show no meaningful differences for Experiment 3 compared to the real-student and other simulation

data.[9] Table 2 has only one notable result for Experiment 3: the effect on the probability of SUCC given a previous QUIT, which was obtained for the modification introduced in Experiment 2, has been eliminated or at least reduced. One explanation that can be offered is that following a QUIT the MUST set grows abnormally large because of the addition of all the skills in the QUIT task except those that were already well-learned. By restricting M to any value less than the maximum possible, the next task will contain many fewer difficult skills and will thus be more likely to be completed with greater ease. However, this explanation is actually superfluous, given the explanation of why the very radical change of setting M equal to 1 did not have the more pervasive effects we might have expected.

As far as we have been able to determine, manipulations of the value of M used by the task-selection procedure are not effective because of limitations due to the available curriculum of tasks. During interactive simulation subsequent to Experiment 3 we have found that in a majority of cases when a task involving precisely M MUST skills is being sought, no such task can be found in the curriculum. Instead, the selection procedure is forced to compromise by selecting a task involving a number of MUST skills greater or less than M, depending on whether M = 1 or M = 3. The average effective value of M seems to be between 1 and 2 in both cases. Subsequently, we have investigated the original algorithm where M is always a maximum and found that here too the average effective value of M is closer to 1 or 2. Of course,

---

[9] The apparent increase in skill requests for Experiment 3 as compared to Experiments 1 and 2 is an artifact of truncating the Experiment 3 sequences. The data for the full sequences (not shown) have skill requests at .08 as in the earlier simulations.

setting M to a specific value has an effect is some cases, but our data clearly show that these cases do not have a very great overall effect on the behavior of the selection procedure or the student.

In order to examine the effects of manipulating M, the BIP-I curriculum would have to be expanded with tasks involving different combinations of skills than exist in the operational version. One possibility that we have not yet pursued is to use the simulation with a pseudo-curriculum consisting only of task identifiers and a description in terms of the skills involved in each task, but no actual problems that could be presented to real students. The pseudo-curriculum could be very large, containing hundreds of tasks involving most of the conceivable combinations of skills, and would enable a selection procedure to find tasks having specified values of M consistently.

General discussion

We have found automated simulation to be a useful supplement to interactive simulation and real-student experiments in our research on individualized task selection in the BIP-I system. In first developing and testing our simulation program with a task-selection procedure for which we had real-student data, we realized that the simulation serves an unanticipated role for analyzing existing data. Our need to pool entries in the simulation database to reduce variability in the simulation's predictions led us to discover that the performance of students in an earlier experiment seems not to have depended to any detectable extent on the number of well-learned skills in the tasks they worked.

Experiment 2 demonstrated that the simulation is sensitive in

reasonable ways to small modifications to the task-selection procedure. Experiment 3 unexpectedly predicted no effects for a conceptually important modification to the selection procedure. Our subsequent analysis revealed that this prediction was in fact sound because the breadth of the current BIP curriculum can limit procedures by preventing them from finding tasks satisfying specific criteria. Because the different procedures have similar default criteria for selecting tasks in such cases, the tasks they select are similar.

The simulation was also useful in finding bugs in the early stages of development of the selection procedures used in Experiments 2 and 3. In the case of making M a parameter to the selection procedure in Experiment 3, the simulation revealed a context-dependent bug that occurred only when M equaled 1 and the procedure was unable to locate a task with only one skill from the MUST set.

Because the simulation did not predict that either of the modified procedures in Experiments 2 and 3 would have any substantial effects, we chose not to test them with a group of real students. To test whether the simulation's predictions are quantitatively sound, we require a selection procedure for which the predicted data are substantially different from the data used to build the simulation database. We did choose to incorporate the modifications tested in Experiments 2 and 3 into the BIP-II system. At the same time, we added more tasks to the curriculum in an attempt to make manipulations of M more effective.

V. Task selection using a network representation of knowledge[10]

Our initial use of a CIN in BIP-I for selecting tasks has demonstrated the successful application of the CIN paradigm (Barr, et al., 1976). However, the representation of programming knowledge and the model of student learning used in BIP-I are very rudimentary. Most obviously, BIP-I's grouping of skills into techniques is an oversimplification of the actual interrelations between skills. The technique groups do not provide a sufficient basis for anticipating a student's performance in new contexts based on his performance in related contexts -- an important aspect of a human tutor's skill in selecting tasks for his student. Likewise, the student model, consisting of counters for each skill, does not differentiate various levels of skill mastery indicated by the amount of difficulty a student encounters in completing tasks. As described in Section IV, our use of simulation indicated that limited modifications to BIP-I would be insufficient to overcome the observed weaknesses in its task-selection behavior. We therefore undertook to design a new CIN-based task-selection procedure for a BIP-II system, incorporating both a more detailed representation for the knowledge underlying the curriculum and more complex assumptions for modeling student learning.

In considering alternative representations for the knowledge underlying a task, we recognized that the most powerful approach would be a procedural representation sufficient to synthesize task solutions (Self, 1974; see, for example, Brown, et al., 1975, and Carr &

---

[10] A version of this section will appear in the Proceedings of the National Association of Computing Machinery, 1977.

Goldstein, 1977). However, the state-of-the-art in program synthesis and analysis techniques has not yet advanced to a point where a manageable system could be implemented for automatically solving programming problems like those in the BIP curriculum. Thus, we decided to extend the original concept of a set of skills by embedding the skills in a network representation describing the structural and pedagogically significant relations between them. The network supersedes the technique groupings. It enables inferences that potentially add sophistication both to the process of task selection and to the interpretation of student performance in updating the student model. For example, unlearned skills that are deemed to be analogous to other skills that are already learned can be given lower priority for inclusion in the next task to be presented. Or, if such skills occur in a task that a student quits, then they can be taken as less likely sources of his difficulty than unlearned skills that are analogous to other skills that are already known to be troublesome for that student.

The BASICNET

Rather than basing the network of knowledge to be learned solely on the BIP curriculum, we built the skill relationships on a general representation for BASIC programs. From an analysis of the BASIC language, guides to BASIC programming, and the skills and techniques of BIP-I, we developed a network representation for BASIC programming constructs (the BASICNET), a simplified portion of which is shown in Figure 10. The node names are self-explanatory; the links (relationships) are Kind, Component, Hardness, and (mutual functional) Dependency. The section of the BASICNET shown specifies that there are

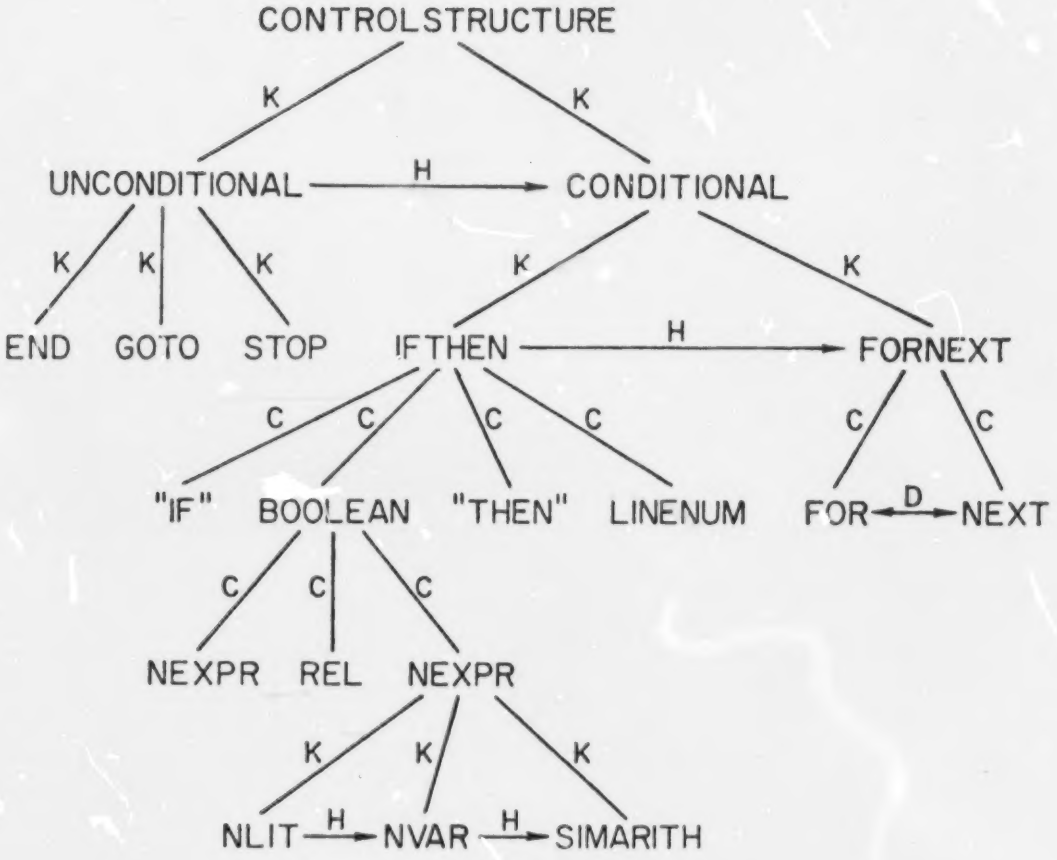

Figure 10. A simplified portion of the BASICNET describing control structure of the BASIC language.

two kinds of control structures, and expresses a judgment that the conditional kind is harder to learn than the unconditional. There are two kinds of conditional structures, and FORNEXT is harder than IFTHEN. The components of an IFTHEN statement are the words "IF" and "THEN" with the Boolean condition and the line number in the appropriate places. For the purposes of this illustration, the BOOLEAN consists of a numeric expression (NEXPR), a relational operator (REL), and another NEXPR; among the three kinds of NEXPRs, numeric literals (NLIT) are easiest, and numeric variables (NVAR) and simple arithmetic expressions (SIMARITH) are increasingly hard.

Note that the downward links in Figure 10 provide information like that found in a BNF notation for BASIC, while the horizontal links provide pedagogical information specifying relative difficulty, analogy, and dependency. The opinions expressed by the horizontal links are necessarily general and do not always hold for all students in all stages of learning. For example, an arithmetic expression is generally a harder construct than a numeric variable because it often includes a variable itself, but observation indicates that using a statement such as PRINT 6+4 tends to be an easier task for a beginner than using PRINT N. This implies that, ultimately, the pedagogical relationships between concepts must sometimes be a function of the student's state of learning at the time the relationships are to be used. We have chosen not to tackle this refinement in the BASICNET underlying the BIP-II system.

List notation for the BASICNET

A simplified version of the list notation we use to represent the portion of the BASICNET in Figure 10 is:

```
(CONTROLSTRUCTURE K (UNCONDITIONAL CONDITIONAL))
(UNCONDITIONAL K (END GOTO STOP) H (CONDITIONAL))
```

```
(CONDITIONAL K (IFTHEN FORNEXT))
(IFTHEN C ("IF" BOOLEAN "THEN" LINENUM) H (FORNEXT))
(FORNEXT C (FOR NEXT))
(FOR D (NEXT))
(BOOLEAN C (NEXPR REL NEXPR))
(NEXPR K (NLIT NVAR SIMARITH))
(NLIT H (NVAR) A (SLIT))
(NVAR H (SIMARITH) A (SVAR) S (SVAR))
```

The A links specify that numeric literals are analogous to string literals (SLIT), and that numeric variables are analogous to string variables (SVAR). The S link says that NVAR and SVAR are similarly difficult. (Note that these A and S relations are not shown in Figure 10. The information about SLIT and SVAR is found in another part of the BASICNET.) The notation here is simply that used to express property lists in LISP. (The list notation for the entire BASICNET is given in Appendix C, and glossary of terminology and notation in Appendix D.)

The BASICNET and BIP skills

After the BASICNET was defined, each skill in BIP's CIN was represented in terms of a subnet.[11] First, the structure of each skill was described, in list notation like the following (where Skill 42 is "conditional branch, comparing a numeric literal with a numeric variable"):

(SK042 (IFTHEN (BOOLEAN . (NEXPR . NLIT) (NEXPR . NVAR))))

Skill 42 is represented as an instance of IFTHEN (see Figure 10), in which the BOOLEAN component is further specified as consisting of the relation between a numeric literal (the first NEXPR component of the BOOLEAN) and a numeric variable (the second NEXPR). The REL is left

---

[11] The development of BIP-II included the definition of 10 additional skills, which are given in Appendix E.

uninstantiated, since Skill 42 does not specify the kind of comparison to be made between the two. Thus any REL is appropriate.

Skill 43 is "conditional branch, comparing a simple numeric expression with a numeric variable." Its structure is

(SK043 (SK042 (NEXPR . SIMARITH)))

The notation is read "Skill 43 is identical to Skill 42 except that the first instance of NEXPR should be SIMARITH," which is exactly what the English description of the skill says. Skill 46 ("conditional branch, comparing two numeric variables") is represented as

(SK046 (SK042 (NEXPR . NVAR)))

again reflecting the minimal difference between the related skills. (The skill structures for BIP-II skills are given in Appendix F.)

Skill sets

Based on the notation for skill structures, skills were grouped together into ten major skill sets representing printing, numeric assignment, string assignment, IF-THENS, FOR-NEXTS, etc. Each skill set was formed by starting with a head skill, not described in terms of any other skill -- like Skill 42 above -- and all other skills (43, 46, etc.) described in terms of it, or described in terms of other members of the set. As might be expected, there was some similarity between the ten skill sets and the technique groupings of BIP-I.

The SKILLSNET

Within each skill set, pairs of skills were examined to find their minimal difference. If the nodes by which they differ are linked in the BASICNET, that link was used to define a relation between the skills. If the nodes by which two skills differ do not have a direct

link, relations were sought at increasingly higher levels of the BASICNET.

For instance, since the BASICNET shows NVAR to be harder than NLIT, and SIMARITH harder than NVAR, it follows that Skill 46 is harder than 42, and 43 is harder than 46. The relationships determined in this manner between all pairs of skills comprise the SKILLSNET, a knowledge representation that can be directly expressed in a CIN and used for task selection. The SKILLSNET, like the BASICNET, can be expressed in LISP property list notation. (The underlining in the following example emphasizes the relationships being discussed here.)

(SK042 H (SK044 SK046 SK047) A (SK047) P (SK003 SK036 SK039))
(SK043 H (SK047 SK075 SK061) P (SK036 SK040 SK003 SK005))
(SK046 H (SK043 SK048 SK045) A (SK048) P (SK003 SK036 SK039))

The P links shown here are Prerequisite links; like the Hardness links, these are a matter of pedagogical opinion. The P links, however, appear only in the SKILLSNET, not in the BASICNET, and express judgments that are more specific to the BIP course than those expressed in the BASICNET. A few of the skills (e.g., those involving the use of built-in BASIC functions such as INT and SQR) did not fall into skill sets since they seemed not to be describable in terms of any other skill. These are related within the SKILLSNET only by means of P links. (The entire SKILLSNET is given in Appendix G).

BIP-II task-selection procedure

The new task-selection procedure for the BIP-II system, designed to use the relationships between skills expressed in the SKILLSNET, is identical to the technique-based method in its overall design: A set of skills appropriate to the student's current level of understanding is

generated, a set of tasks using some of those skills is identified, the best of those tasks (by some criteria) is presented, and the student model is updated based on the student's performance and self-evaluation on the task. The major difference between the two methods is that by using the expanded set of relations between skills in the SKILLSNET, the new procedure can make more intelligent inferences both in updating the student model and in generating the set of skills to be involved in the student's next task.

BIP-II incorporates a finite-state model of learning with five possible states. The states are:

UNSEEN	Not yet seen in a task (not learned)
TROUBLE	Required by a task, but not learned
MARGINAL	Learned to a marginal degree
EASY	Not yet seen, but probably easy to learn
LEARNED	Learned to a sufficient degree

Skills can move to the TROUBLE, MARGINAL, and LEARNED states as a function of the difficulty a student has in completing tasks that involve them. A skill can become EASY if a skill that is harder than it becomes LEARNED. UNSEEN and EASY skills can also become LEARNED, MARGINAL, or TROUBLE if they are prerequisites of skills that move into those states.

Figure 11 is a simplified description of the process by which a task is selected at any point during instruction, given the student's state of knowledge of all the skills. The procedure integrates a number of a priori reasonable pedagogical heuristics about how to vary the relative difficulty of tasks to optimize learning as performance varies and about how to teach a network of knowledge (e.g., breadth-first vs. depth-first exposure).

As an example of the inferences made in the generation of the

STEP 1: Create a set called NEED, consisting of skills that will be sought in the next task. Look for "trouble" skills first (those in tasks that the student quit), then for analogies to learned skills, then for inverse-prerequisites of learned skills. As soon as a group of such skills is found, stop looking.

STEP 2: Remove from the NEED set those skills that have unlearned prerequisites. Add those skills to the NOTREADY set (which may be used later).

STEP 3: Given a NEED set, find the most appropriate task that involves some of the NEEDed skills.

(a) Assemble GOODLIST, those tasks that have the desired number of NEEDed skills. (This number increases if the student is consistently successful, decreases if he has trouble.)

(b) If no GOODLIST can be created, make a new NEED set consisting of the prerequisites of the skills in NOTREADY. If no new NEED set can be created, then the curriculum has been exhausted; otherwise GOTO 3a.

(c) Find the "best" task: if the student is doing well, find the task in GOODLIST that has the fewest learned skills; if he is progressing more slowly, find the task with the fewest unseen skills. Remove the selected task from GOODLIST.

STEP 4: See if the selected task is otherwise appropriate.

(a) If none of the skills in the selected task have unlearned prerequisites, stop looking and present the task. (END)

(b) If any skills have unsatisfied prerequisites, reject the task and add those skills to the NOTREADY set.

(c) If GOODLIST is exhausted, change (usually reduce) the criterial number of NEED skills, and GOTO 3a. Otherwise, using the rest of GOODLIST GOTO 3c.

Figure 11. Outline of BIP-II task-selection process.

NEED set, let us assume that skills 42 and 61 ("FOR . NEXT loops with literal as final value of index") are under consideration. Skill 61 is represented as

(SK061 H (SK062) P (SK042))

The Prerequisite relationship specifies that 42 must be learned before a task involving 61 can be presented. The structure enforced by the P links relating pairs of skills gives the presentation of tasks some degree of order, and is designed to prevent too-rapid progress or drastic jumps in difficulty-- in this way, they function like the BIP-I technique ordering. The Hardness links, in contrast, are used to facilitate progress for a student doing well, by allowing some skills to be considered "too easy" for inclusion in the NEED set. Such skills are not inferred to be learned; they are simply not sought actively by the selection algorithm.

As an example using the skills described here, suppose that a student has successfully completed a task using Skill 43, although he has not yet seen Skill 42. (The fact that 43 is harder than 42 does not force 42 to be presented first; only P links force such order.) When the task-selection procedure assembles the next set of NEED skills, it will "infer" that 42 is now too easy to become part of that set, since something harder than 42 has already been learned.

Furthermore, since 42 is now considered too easy to look for, Skill 61 can now be sought. If the student successfully completes a task involving 61, the student model will be updated to show that 61 has been learned, and by inference, that its unseen prerequisite 42 has also been learned. These kinds of inferences (by which skills can reach

too-easy or learned states without the student actually having seen them in a task) can of course be contradicted by direct observation or by other inferences if the student has difficulty. For example, the unseen prerequisite of a given skill may change its state from EASY to TROUBLE if the student quits (gives up on) a task involving the given skill. The next task selection would attempt to find a task using that prerequisite skill in such a case.

BIP-II performance

The BIP-II task-selection procedure has been implemented with parameters (e.g, numbers of skills sought and thresholds determining when an unpresented skill is "too easy" to be included in the NEED set) that can be changed readily. The system can therefore be used to explore the effectiveness of somewhat different pedagogical heuristics for task sequencing. We used this capability in conjunction with an interactive simulation system to create a version of BIP-II that we expected would be effective for a range of student abilities. Recently, we collected data from a group of 28 students who used this BIP-II system.

The students were limited to fifteen hours of terminal time with BIP. They were presented with (but did not necessarily complete successfully) an average of 40 tasks, the minimum being 21 tasks and the maximum 63 tasks. Only one student dropped out of the course without completing fifteen hours or finishing the curriculum.

One overall measure of the success of the semantic network CIN and related task-selection procedure is the relationship between number of skills learned (according to BIP) and scores on a paper-and-pencil

posttest. The correlation between these measures was .86, and accounts for 74 percent of the variability in the posttest scores. The students also took a standardized test of programming aptitude prior to instruction. The pretest scores correlated significantly with both the number of skills learned (r = .65) and the posttest scores (r = .59); nowever, in a multiple regression of the posttest scores on the number of skills learned and the pretest scores, only the number of skills learned contributed significantly to posttest performance. Number of tasks presented to students was independent of both test scores and number of skills learned. Thus, the student model maintained by the BIP-II system accurately reflects the acquisition of the programming knowledge required by the posttest, and predicts posttest performance independently of the aptitude measured by the pretest.

Our informal observations indicate that BIP-II task sequences are substantially different from those of BIP-I. Most noticeably, when a student does well initially, BIP-II selects complex tasks at a much earlier point than they were selected by BIP-I. Students' reactions were favorable, even when they spent considerable time on these difficult tasks and then gave up. Task selection following these failures seems responsive: BIP-II selected simpler tasks involving some "not-learned" skills that were involved in the task that the student had quit. In many cases, these "remedial" tasks appear to have been too simple, given the student's prior progress, but most often BIP-II was in fact looking for a more challenging task and could not find one in the curriculum (i.e., the tasks added to the curriculum were insufficient to relieve this problem, first observed in BIP-I-- see Section IV).

Besides the sequencing "failures" due to the limits of the

curriculum, a large number of inadequate task-selection decisions were caused by poor data from the system's solution checker. In these cases, the student had a substantially correct program that was rejected by the checker. BIP-II interprets the rejections as a sign that the student is having difficulty with some of the skills in the task and thereby introduced errors into the student model. Sometimes students in our study became so frustrated by the rejection of their programs that they quit the task, creating even more severe errors in the model.

Disregarding the difficulties caused by these weaknesses elsewhere in instructional system, our initial evaluation of BIP-II's task-selection capabilities is favorable. However, it will be difficult to evaluate of the effects of manipulating the parameters of the selection process until the curriculum is substantially expanded and the solution checker is improved.

VI. Concluding remarks

We conclude this report with summary remarks about the automated simulation procedure and network-based CIN structure developed during the present research.

Our conclusion for the present regarding the use of automated simulation with CAI systems is that it can reduce the amount of expensive student testing required to evaluate modifications to a system. The simulation enables detection of unanticipated problems (as opposed to the anticipated problems that can be examined with interactive simulation), and so the CAI system is likely to be in a more robust state by the time real students use it. Predictions from a simulation about the effectiveness of modifications to an instructional system can be judged subjectively and used to determine whether a modification is promising enough to be fully implemented and evaluated with real students.

BIP-II indicates how complex knowledge representations adapted from AI research can be used to describe a CAI problem curriculum and thereby enable relatively sophisticated individualized problem sequencing. In particular, a CIN based on a semantic network representation provides a medium for drawing indirect inferences about what a student knows, what he is ready to learn, and what task in the curriculum will best help him learn it. A network representation then is useful not only for expressing unambiguous relationships (e.g., property-inheritance, componency), as it is typically used in AI systems, but, in addition, allows one to express systematically certain opinions about the pedagogical relationships among the concepts and skills a CAI system is intended to teach.

We do not believe that the BIP-II system, based on a CIN incorporating a complex network of skill relationships, can match a human tutor's ability to select programming problems adaptively: The limitations imposed by the system's rudimentary program checker insure some extreme failures, but, beyond this, the SKILLSNET and the inferences that use it still only weakly approximate the flexibility of which a tutor is capable. Nonetheless, the more evolved CIN makes it possible for tutorial CAI in technical subjects to individualize student experience effectively across a range of student abilities and instructional objectives.

References

Anderson, R.H., & Gillogly, J.J. Rand Intelligent Terminal Agent (RITA): Design philosophy. Report R-1809-ARPA, The Rand Corporation, Santa Monica, California, 1976.

Atkinson, R. C., Fletcher, J. D., Lindsay, E. J., Campbell, J. O., & Barr, A. Computer-assisted instruction in initial reading. Technical Report 207. Institute for Mathematical Studies in the Social Sciences, Stanford University, Stanford, California, July, 1973.

Barr, A., Beard, M., & Atkinson, R. C. A rationale and description of a CAI program to teach the BASIC programming language. Instructional Science, 1975 4, 1-31. (a)

Barr, A., Beard, M., & Atkinson, R.C. The computer as a tutorial laboratory: The Stanford BIP Project. International Journal of Man-Machine Studies, 1976, 8, 567-596.

Beard, M., Lorton, P., Searle, B., & Atkinson, R.C. Comparison of student performance and attitude under three lesson-selection strategies in computer-assisted instruction. Technical Report 222. Institute for Mathematical Studies in the Social Sciences, Stanford University, Stanford, California, December, 1973.

Bork, A. Learning with computers -- today and tomorrow. In O. Lecarme and P. Lewis, Eds. Computers in education (IFIP 2nd World Conference). Amsterdam: North Holland, 1975.

Brown, J.S. & Burton, R.R. Multiple representations of knowledge for tutorial reasoning. In D.G. Bobrow and A. Collins (Eds.) Representation and understanding: Studies in cognitive science. New York: Academic Press, 1975.

Brown, J.S., Burton, R.R., Hausmann, C., Goldstein, I., Huggins, B., & Miller, M. Aspects of a Theory for automated student modelling. BBN Report No. 3549, Bolt Beranek and Newman, Inc., Cambridge, Mass., May, 1977.

Brown, J.S., Burton, R.R., Miller, M., deKleer, J., Purcell, S., Hausmann, C., & Bobrow, R. Steps toward a theoretical foundation for complex, knowledge-based CAI. BBN Report No. 3135, Bolt Beranek and Newman, Inc., Cambridge, Mass., August, 1975.

Carbonell, J. R. AI in CAI: An artificial intelligence approach to computer-aided instruction. IEEE Transctions on Man-Machine Systems, 1970, MMS-11, 190-202.

Carr, B., & Goldstein, I.P. Overlays: a theory of modelling for computer aided instruction. MIT AI Memo 406, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, Mass., February, 1977.

Collins, A.M. Processes in acquiring knowledge. In R.C. Anderson, R.J. Spiro, & W.E. Montague (Eds.), Schooling and the acquisition of knowledge. Hillsdale, N.J.: Lawrence Erlbaum Associates, 1977.

Collins, A.M., & Grignetti, M. Intelligent CAI. BBN Report 3181, Bolt Beranek and Newman, Inc., Cambridge, Mass., October, 1975.

Collins, A.M., Warnock, E.H., & Passafiume, J.J. Analysis and synthesis of tutorial dialogues. In G. Bower (Ed.), The psychology of learning and motivation (Vol. 9). New York: Academic Press, 1975.

Danforth, D. G., Rogosa, D. R., & Suppes, P. Learning models for real-time speech recognition. Technical Report 223, Institute for Mathematical Studies in the Social Sciences, Stanford University, Stanford, California, January, 1974.

Friend, J. Computer-assisted instruction in programming: A curriculum description. Technical Report 211, Institute for Mathematical Studies in the Social Sciences, Stanford University, Stanford, California, July, 1973.

Friend, J. Programs students write. Technical Report 257, Institute for Mathematical Studies in the Social Sciences, Stanford University, Stanford, California, July, 1975.

Goldberg, A. Computer-assisted instruction: The application of theorem-proving to adaptive response analysis. Technical Report 203, Institute for Mathematical Studies in the Social Sciences, Stanford University, Stanford, California, May, 1973.

Goldstein, I. Summary of MYCROFT: A system for understanding simple picture programs. Artificial Intelligence, 1975, 6, 249-288.

Grignetti, M. C., Hausmann, C., & Gould, L. An "intelligent" on- line assistant and tutor: NLS-SCHOLAR. Proceedings of the National Computer Conference, Anaheim, Ca., 1975, 775-781.

Groen, G.J., & Atkinson, R.C. Models for optimizing the learning process. Psychological Bulletin, 1966, 4, 309-320.

Kimball, R. B. Self-optimizing computer-assisted tutoring: Theory and practice. Technical Report 206, Institute for Mathematical Studies in the Social Sciences, Stanford University, Stanford, California, June, 1973.

Koffman, E. B. A generative CAI tutor for computer science concepts. Proceedings AFIPS 1972 Spring Joint Computer Conference, 1972, 379-389.

Koffman, E. B., & Blount, S. E. Artificial intelligence and automatic programming in CAI. Artificial Intelligence, 1975, 6, 215-234.

*Reiser, J.F. BAIL-- A debugger for SAIL. Memo AIM-270, Stanford Artificial Intelligence Laboratory, Stanford, California, October, 1975.

Reiser, J.F. (Ed.) SAIL. Memo AIM-289, Stanford Artificial Intelligence Laboratory, Stanford, California, August, 1976.

Ruth, G. Analysis of algorithm implementations. MAC TR-130, Massachusetts Institute of Technology, Cambridge, Mass., May, 1974.

Sanders, W., Benbassat, G., & Smith, R. L. Speech synthesis for computer-assisted instruction: The MISS system and its applications. SIGSCE Bulletin, 1976, 8, 200-211.

Self, J.A. Student models in computer-aided instruction. International Journal of Man-Machine Studies, 1974 6, 261-276.

Smith, R. L., & Blaine, L. A generalized system for university mathematics instruction. SIGSCE Bulletin, 1976, 8, 280-208.

Smith, R. L., Graves, H., Blaine, L. H., & Marinov, V. G. Computer-assisted axiomatic mathematics: Informal rigor. In O. Lecarme and R. Lewis, Eds., Computers in Education (IFIP 2nd World Conference). Amsterdam: North Holland, 1975.

Suppes, P., & Morningstar, M. Computer-assisted instruction at Stanford, 1966-68: Data, models, and evaluation of the arithmetic programs. New York: Academic Press, 1972.

Van Campen, J. A. Project for application of learning theory to problems of second language acquisition with particular reference to Russian. Report to U. S. Office of Education, Contract No. OEC-0-8-001209-1306, 1970.

Wollmer, R.D., & Bond, N.A. Evaluation of a Markov-decision model for instructional sequence optimization. Technical Report No. 76, Dept. of Psychology, Univ. of Southern California, Los Angeles, California, October, 1975.

Appendix A. BIP-I Technique Groups and Skills

Technique 1
1 Print numeric literal
2 Print string literal
5 Print numeric expression [operation on literals]
8 Print string expression [concatenation of literals]

Technique 2
3 Print value of numeric variable
4 Print value of string variable
6 Print numeric expression [operation on variables]
7 Print numeric expression [operation on literals and variables]
9 Print string expression [concatenation of variables]
10 Print string expression [concatenation of variable and literal]
11 Assign value to a numeric variable [literal value]
12 Assign value to a string variable [literal value]

Technique 3
34 Assign to a string variable [value of an expression]
35 Assign to a numeric variable [value of an expression]
69 Re-assignment of string variable (using its own value)
70 Re-assignment of numeric varable (usin its own value)
82 Assign to numeric variable the value of another variable
83 Assign to string variable the value of another variable

Technique 4
28 Multiple print [string literal, numeric variable]
29 Multiple print [string literal, numeric variable expression]
30 Multiple print [string literal, string variable]
74 Multiple print [string literal, string variable expression]

Technique 5
13 Assign numeric variable by -INPUT-
14 Assign string variable by -INPUT-
15 Assign numeric variable by -READ- and -DATA-
16 Assign string variable by -READ- and -DATA-
55 The REM statement

Technique 6
17 Multiple values in -DATA- [all numeric]
18 Multiple values in -DATA- [all string]
19 Multiple values in -DATA- [mixed numeric and string]
22 Multiple assignment by -INPUT- [numeric variables]
23 Multiple assignment by -INPUT- [string variables]
24 Multiple assignment by -INPUT- [mixed numeric and string]
25 Multiple assignment by -READ- [numeric]
26 Multiple assignment by -READ- [string]
27 Multiple assignment by -READ- [mixed numeric and string]

Technique 7
36 Unconditional branch (-GOTO-)
37 Interrupt execution

Technique 8
38 Print Boolean expression [relation of string literals]
39 Print Boolean expression [relation of numeric literals]
40 Print Boolean expression [relation of numeric literal and variable]
41 Print Boolean expression [relation of string literal and variable]
75 Boolean operator -AND-
76 Boolean operator -OR-
77 Boolean operator -NOT-

Technique 9
42 Conditional branch [compare numeric variable with numeric literal]
43 Conditional branch [compare numeric variable with expression]
46 Conditional branch [compare two numeric variables]
47 Conditional branch [compare string variable with string literal]
48 Conditional branch [compare two string variables]
59 The -STOP- statement

Technique 10
44 Conditional branch [compare counter with numeric literal]
45 Conditional branch [compare counter with numeric variable]
49 Initialize counter variable with a literal value
50 Initialize counter variable with the value of a variable
53 Increment the value of a counter variable
54 Decrement the value of a counter variable

Technique 11
51 Accumulate successive values into numeric variable
52 Accumulate successive values into string variable
71 Calculating complex expressions [numeric literal and variable]
78 Initialize numeric variable (not counter) to literal value
79 Initialize numeric variable (not counter) to value of a variable
80 Initialize string variable to literal value
81 Initialize string variable to the value of another variable

Technique 12
20 Dummy value in -DATA- statement [numeric]
21 Dummy value in -DATA- statement [string]

Technique 13
56 The -INT- function
57 The -RND- function
58 The -SQR- function

Technique 14
61 FOR . NEXT loops with literal as final value of index
62 FOR . NEXT loops with variable as final value of index
63 FOR . NEXT loops with positive step size other than 1
64 FOR . NEXT loops with negative step size

Technique 15
31 Assign element of string array variable by -INPUT-
32 Assign element of numeric array variable by -INPUT-
33 Assign element of numeric array variable [value is also a variable]
60 The -DIM- statement
65 String array using numeric variable as index
66 Print value of an element of a string array variable
67 Numeric array using numeric variable as index
68 Print value of an element of a numeric array variable

Technique 16
72 Nesting loops
73 Subroutines (-GOSUB- and friends)

Appendix B. Definition of R_n

Given two sets A and B each containing any numbers of task sequences, it is desired to find some measure of how close these sets are to each other. Let mA(i) be the number of times the ith sequence of length n (say n = 2) occurred in set A, and let mB(i) be the number of times the ith sequence occurred in set B (where i indexes all possible sequences of length n). We compute R_n, the "dot product" of A and B (denoted by A . B), as follows:

$$R_n = A \cdot B = MAB \;/\; \mathrm{sqrt}(\, MAA * MBB)$$

where MAB denotes the sum over all possible i of mA(i) * mB(i), MAA denotes the sum over all possible i of mA(i) * mA(i), and MBB denotes the sum over all possible i of mB(i) * mB(i). The motivation for this formula is provided by a vector analogy: if A and B are viewed as vectors in a multi-dimensional space, and the mA(i) and mB(i) are regarded as the components of these respective vectors, then the dot (scalar) product provides an indication of how close these vectors are to being in the same direction. The cosine of the angle between these vectors is in fact given by the dot product of the normalized vectors, and is precisely the expression given above for A . B; since the mA(i) and mB(i) cannot be negative, A . B can range in value from zero to one, the former representing an angle of 90 degrees (mutually perpendicular) and the latter representing an angle of zero degrees (complete coincidence). Note if the frequencies of n-sequences in A and B are completely identical, A . B will have a value of one, whereas if no n-sequences in A occurs at all in B and vice versa, A . B will have a value of zero. The higher the value of A . B the more similar the

sequences of tasks. If both $\underline{A}$ and $\underline{B}$ are samples from the same parent population, then the expected value of $\underline{A} \cdot \underline{B}$ approaches one as the sample size increases without bound; for a finite sample size, the expected value of $\underline{A} \cdot \underline{B}$ is somewhat less than one, the exact value depending on $\underline{n}$, on the number of elements in the sample, on the total number of tasks, etc. Note that the expression for $\underline{A} \cdot \underline{B}$ looks formally like the expression for the product-moment correlation of two variables.

Appendix C. LISP Notation for the BASICNET

```
(BASICPROGRAM C ((STATEMENTLINES . OPT) ENDSTATEMENT))
(STATEMENTLINES C (LINENUM (STATEMENTS . OPT)))
(ENDSTATEMENT C (LINENUM ENDST))
(STATEMENTS K (REMST IOST ASSIGNST CONTROLST DIMST))
(ENDST C (%END%))
(REMST C (%REM% (TEXT . OPT)))
(IOST K (PRINTST INST))
(INST H (PRINTST LETST) C (INNAME VARLIST) K (INPUTST READATAST))
(ASSIGNST K (INST LETST))
(DIMST C (%DIM% VAR NEXPR) D (SUBVAR))
(CONTROLST K (UNCONDITIONAL CONDITIONAL))
(CONDITIONAL H (UNCONDITIONAL) K (IFTHENST FORNEXTST))
(UNCONDITIONAL K (ENDST GOTOST STOPST))
(STOPST H (GOTOST) C (%STOP%))
(FORST D (NEXTST)
       C (%FOR% (NUMVAR . 1) %FROM% NEXPR %TO% NEXPR STEPEXPR))
(GOTOST C (%GOTO% LINENUM) H (ENDST))
(LETST K (NUMLET STRLET) C ((%LET% . OPT) VARIABLE GETS EXPR))
(STEPEXPR K (NOSTEP STEP))
(FORNEXTST H (IFTHENST) C (FORST NEXTST))
(NEXTST D (FORST) C (%NEXT% (NUMVAR . 1)))
(NUMLET C ((%LET% . OPT) NUMVAR GETS NEXPR))
(STRLET C ((%LET% . OPT) STRVAR GETS SEXPR) H (NUMLET) A (NUMLET))
(NOSTEP C (NILL))
(STEP C (%STEP% NEXPR) H (NOSTEP))
(IFTHENST C (%IF% BEXPR %THEN% LINENUM))
(PRINTST C (%PRINT% EXPRLIST))
(EXPRLIST K (EXPR EXPRS))
(EXPRS H (EXPR) C (EXPR EXPRLIST))
(EXPR K (SIMPLEXPR COMPLEXPR NEXPR SEXPR BEXPR))
(SIMPLEXPR K (SIMNEXPR SIMSEXPR))
(COMPLEXPR H (SIMPLEXPR) C (SIMPLEXPR OPERATOR EXPR))
(SIMSEXPR H (SIMNEXPR) K (SLIT STRVAR) A (SIMNEXPR))
(SEXPR K (SIMSEXPR CONCATSEXPR FUNCSEXPR) H (NEXPR))
(CONCATSEXPR C (SIMSEXPR CONCAT SEXPR) H (SIMSEXPR) A (SIMARITH))
(SIMNEXPR K (NLIT NUMVAR) A (SIMSEXPR))
(NEXPR K (SIMNEXPR ARITHNEXPR FUNCNEXPR))
(ARITHNEXPR K (SIMARITH COMPLARITH) H (SIMNEXPR))
(SIMARITH C (SIMNEXPR ARITH SIMNEXPR) A (CONCATSEXPR))
(COMPLARITH C (SIMARITH ARITH NEXPR) H (SIMARITH))
(FUNCNEXPR K (SQR INT RND) H (SIMNEXPR FUNCSEXPR))
(SQR C (%SQR% LPAREN NEXPR RPAREN))
(INT C (%INT% LPAREN NEXPR RPAREN) H (SQR) S (RND))
(RND C (%RND%) S (INT))
(BEXPR K (SIMREL BOOLEREL) H (SEXPR))
(BOOLEREL C (BEXPR BOOLOP BEXPR) H (SIMREL))
(SIMREL K (NREL SREL))
(SREL C (SEXPR RELOP SEXPR) H (NREL))
(NREL C (NEXPR RELOP NEXPR))
(INNAME K (%READ% %INPUT%))
(INPUTST C (%INPUT% VARLIST))
```

```
(VARLIST K (VARIABLE VARIABLES))
(VARIABLES H (VARIABLE) C (VARIABLE VARLIST))
(READATAST C (READST DATAST) H (INPUTST))
(READST C (%READ% VARLIST) D (DATAST) S (DATAST))
(DATAST C (%DATA% LITLIST) D (READST) S (READST))
(LITLIST K (LIT LITS))
(LITS C (LIT LITLIST) H (LIT))
(OPERATOR K (ARITH CONCAT RELOP BOOLOP))
(ARITH K (ADDSUB MULTDIV EXP))
(EXP H (MULTDIV) C (%/%))
(MULTDIV H (ADDSUB) K (MULT DIV))
(DIV H (MULT) C (%//%))
(MULT C (%/*%))
(ADDSUB K (ADD SUB))
(SUB H (ADD) C (%/-%))
(ADD C (%/+%) A (CONCAT))
(CONCAT C (%/&%) A (ADD) H (ADD))
(BOOLOP K (AND OR NOT) H (RELOP))
(OR H (AND) C (%OR%))
(AND C (%AND%))
(TEXT C (CHARACTER (TEXT . OPT)))
(RELOP K (EQUAL COMP EQCOMP))
(EQCOMP H (COMP) K (GE LE))
(LE S (GE) C (%/</=%))
(GE S (LE) C (%/>/=%))
(COMP K (GT LT) H (EQUAL))
(GT S (LT) C (%/>%))
(LT S (GT) C (%/<%))
(EQUAL K (EQ NEQ))
(NEQ H (EQ) C (%/>/<%))
(EQ C (%/=%))
(VARIABLE K (VAR SUBVAR NUMVAR STRVAR) H (LITERAL))
(SUBVAR D (DIMST) K (SUBNVAR SUBSVAR) H (VAR) C (VAR INDEX))
(VAR K (NVAR SVAR))
(SVAR A (NVAR) S (NVAR))
(NVAR A (SVAR) S (SVAR))
(SUBSVAR S (SUBNVAR) A (SUBNVAR) C (SVAR INDEX) H (SVAR))
(SUBNVAR C (NVAR INDEX) S (SUBSVAR) A (SUBSVAR) H (NVAR))
(INDEX C (%/(% NEXPR %/)%))
(NUMVAR K (NVAR SUBNVAR) S (STRVAR) A (STRVAR) H (NLIT))
(STRVAR K (SVAR SUBSVAR) S (NUMVAR) A (NUMVAR) H (SLIT))
(LITERAL K (NLIT SLIT))
(SLIT C (QUOTE TEXT QUOTE) H (NLIT) A (NLIT))
(NULL H (SPACE) A (ZERO))
(SPACE H (SYMBOLS) C (%/ %))
(CHARACTER K (LETTER DIGIT SYMBOL SPACE NULL))
(SYMBOL H (DIGIT))
(DIGIT H (LETTER) X (POS) K (%0% %1% %2% %3% %4% %5% %6% %7% %8% %9%))
(NLIT A (SLIT) K (INTEGER REAL))
(REAL H (INTEGER))
(INTEGER K (POS NEG ZERO))
(NEG H (ZERO) C (%/-% DIGITS))
(DIGITS C (DIGIT (DIGITS . OPT)))
(POS X (DIGIT) K (ONE NOTONE) C ((%+% . OPT) DIGITS))
(ZERO H (POS) A (NULL) C (%0%))
```

```
(NOTONE H (ONE))
(ONE C (%1%))
(NOT H (OR) C (%NOT%))
(FUNC K (FUNCNEXPR FUNCSEXPR))
(FUNCSEXPR H (CONCATSEXPR) K (LEN SUBSTRINGSEXPR))
(LEN C (%LEN% LPAREN SEXPR RPAREN))
(SUBSTRINGSEXPR C (SEXPR SUBSTRING) H (LEN))
(SUBSTRING C (LPAREN NEXPR COMMA NEXPR RPAREN))
```

Note: For obvious cases (e.g., LETTER), branches of the network have not been expanded to terminal components. See Appendix D for a glossary of terms and notation.

Appendix D. BASICNET Glossary.

Nodes

ADD	addition operator
ADDSUB	additional and subtraction operators
ARITH	arithmetic operator
ARITHNEXPR	arithmetic expression
ASSIGNST	assignment statement
BASICPROGRAM	BASIC program
BEXPR	Boolean expression, either simple or complex
BOOLEREL	complex Boolean expression with Boolean operator(s)
BOOLOP	Boolean operator (AND. OR, or NOT)
CHARACTER	any keyboard symbol known to BASIC
COMP	GT and LT
COMPLARITH	complex arithmetic expression
COMPLEXPR	complex expression (operator - either string or numeric)
CONCAT	concatenation operator
CONCATSEXPR	string expression with concatenation operator(s)
CONDITIONAL	conditional branching
CONTROLST	control statement
DATAST	DATA statement
DIGIT	0 through 9
DIGITS	one or more DIGIT
DIMST	dimension statement
DIV	division operator
ENDST	END statement
ENDSTATEMENT	the END statement, including its line number
EQCOMP	LE and GE
EQUAL	EQ and NEQ
EXP	exponentiation operator
EXPR	expression (either string or numeric)
EXPRLIST	expression list
EXPRS	multiple expressions (in a PRINT statement)
FORNEXTST	combination of FOR statement and NEXT statement
FORST	FOR statement
FUNCNEXPR	function
GOTOST	GOTO statement
IFTHENST	IF-THEN statement
INDEX	index part of a subscripted variable
INNAME	name of an in statement - either INPUT or READ
INPUTST	INPUT statement
INST	in statement
INT	integer function
INTEGER	integer number (positive, negative, or zero)
IOST	I/O statement
LETST	LET statement
LITERAL	literal (either string or numeric)
LITLIST	literal list (for use with DATA statement)
LITS	multiple literals (for use with DATA statement)
MULT	multiplicaton operator

MULTDIV	multiplication and division operators
NEG	negative integer
NEXPR	numeric expression
NEXTST	NEXT statement
NLIT	numeric literal
NOSTEP	nil (missing) step expression (in a FOR statement)
NOTONE	positive integer, not one
NREL	simple Boolean expression relating numeric expressions
NUMLET	numeric LET statement
NUMVAR	numeric variable (either simple or subscripted)
NVAR	simple numeric variable
OPERATOR	operator - either arithmetic, string, or Boolean
POS	positive integer
PRINTST	PRINT statement
READATAST	combination of READ statement and DATA statement
READST	READ statement
REAL	floating point number
RELOP	relational operator - EQ, NEQ, GT, LT, LE, GE
REMST	remark statement
RND	random number function
SEXPR	string expression
SIMARITH	simple arithmetic expression
SIMNEXPR	simple numeric expression
SIMPLEXPR	simple expression (no operator - either string or numeric)
SIMREL	simple Boolean expression with one relational operator
SIMSEXPR	simple string expression
SLIT	string literal
SQR	square root function
SREL	simple Boolean expression relating string expressions
STATEMENTLINES	BASIC statements, including line numbers
STATEMENTS	BASIC statements, exclusive of line numbers
STEP	overt step expression (in a FOR statement)
STEPEXPR	step expression (in a FOR statement - may be nil)
STOPST	STOP statement
STRLET	string LET statement
STRVAR	string variable (either simple or subscripted)
SUB	subtraction operator
SUBNVAR	subscripted numeric variable
SUBSVAR	subscripted string variable
SUBVAR	subscripted variable (either string or numeric)
SVAR	simple string variable
SYMBOL	CHARACTER which is not a letter, digit, or a space
TEXT	arbitrary text (as in a REMARK statement)
UNCONDITIONAL	unconditional branching
VAR	simple variable (either string or numeric)
VARIABLE	all variables - string and numeric, simple and subscripted
VARIABLES	multiple variables (for use with INPUT or READ

statement)

VARLIST — variable list (for use with INPUT or READ statement)

Relations

C	Components of
K	Kinds of
A	conceptually Analogous to
X	morphologically similar to, but conceptually different from
S	Similarly difficult to use or learn
H	Harder to use or learn than
D	functionally Dependent on

Notation

(1) Nodes surrounded by percent symbols are terminal components.

(2) Components dotted with OPT are optional.

(3) Components which are dotted with an integer must be instantiated identically in any specific expansion down from a node (e.g., the numeric variable used as an index in a FOR-NEXT construction is identical in the FOR statement and the NEXT statement).

(4) Components preceded by a slash ("/") are terminal nodes that are special characters.

Appendix E. Skills Added to BIP-II CIN

84 assign element of numeric array variable with -LET-
85 assign element of string array variable with -LET-
86 assign element of numeric array variable with -READ-
87 assign element of string array variable with -READ-
88 specify substring of a string variable, using numeric literals as pointers
89 specify substring of a string variable, using numeric variables as pointers
90 specify substring of a string variable, using variable and literal as pointers
91 the -LEN- function
92 initialize string variable to the null string
93 multiple print [mixed string and numeric, literals and variables]

Appendix F. LISP Notation for Skill Structures

```
(SK001 (PRINTST (EXPRLIST . NLIT]
(SK002 (SK001 (EXPRLIST . SLIT]
(SK003 (SK001 (EXPRLIST . NVAR]
(SK004 (SK001 (EXPRLIST . SVAR]
(SK005 (SK001 (EXPRLIST . (SIMARITH (SIMNEXPR . NLIT)(SIMNEXPR . NLIT]
(SK006 (SK005 (EXPRLIST . (SIMARITH (SIMNEXPR . NVAR)(SIMNEXPR . NVAR]
(SK007 (SK006 (EXPRLIST . (SIMARITH (SIMNEXPR . NLIT]
(SK008 (SK001 (EXPRLIST . (CONCATSEXPR (SEXPR . SLIT)(SEXPR . SLIT]
(SK009 (SK008 (SEXPR . SVAR)(SEXPR . SVAR]
(SK010 (SK009 (SEXPR . SLIT]
(SK011 (NUMLET (NUMVAR . NVAR)(NEXPR . NLIT]
(SK012 (STRLET (STRVAR . SVAR)(SEXPR . SLIT]
(SK013 (INPUTST (VARLIST . NVAR]
(SK014 (SK013 (VARLIST . SVAR]
(SK015 (READATAST (READST (VARLIST . NVAR))(DATAST (LITLIST . NLIT]
(SK016 (SK015 (READST (VARLIST . SVAR))(DATAST (LITLIST . SLIT]
(SK017 (DATAST (LITLIST . (LITS (LIT . NLIT)(LITLIST . NLIT]
(SK018 (SK017 (LIT . SLIT)(LITLIST . SLIT]
(SK019 (SK018 (LIT . NLIT]
(SK020 (SK017]
(SK021 (SK018]
(SK022 (INST (INNAME . %INPUT%)
             (VARLIST . (VARIABLES (VARIABLE . NVAR)(VARLIST . NVAR]
(SK023 (SK022 (VARIABLE . SVAR)(VARLIST . SVAR]
(SK024 (SK023 (VARIABLE . NVAR]
(SK025 (SK022 (INNAME . %READ%]
(SK026 (SK023 (INNAME . %READ%]
(SK027 (SK024 (INNAME . %READ%]
(SK028 (SK001 (EXPRLIST . (EXPRS (EXPR . SLIT)(EXPRLIST . NVAR]
(SK029 (SK028 (EXPR . SLIT)(EXPRLIST . SIMARITH]
(SK030 (SK028 (EXPR . SLIT)(EXPRLIST . SVAR]
(SK031 (SK013 (VARLIST . SUBSVAR]
(SK032 (SK013 (VARLIST . SUBNVAR]
(SK033 (SK011 (NUMVAR . SUBNVAR)(NEXPR . NVAR]
(SK034 (SK012 (SEXPR . CONCATSEXPR]
(SK035 (SK011 (NEXPR . SIMARITH]
(SK036 (GOTOST]
(SK037 (CTRLG]
(SK038 (SK001 (EXPRLIST . (SREL (SEXPR . SLIT)(SEXPR . SLIT]
(SK039 (SK038 (EXPRLIST . (NREL (NEXPR . NLIT)(NEXPR . NLIT]
(SK040 (SK039 (NEXPR . NVAR]
(SK041 (SK038 (SEXPR . SVAR]
(SK042 (IFTHENST (BEXPR . (NREL (NEXPR . NLIT)(NEXPR . NVAR]
(SK043 (SK042 (NEXPR . SIMARITH]
(SK044 (SK042]
(SK045 (SK046]
(SK046 (SK042 (NEXPR . NVAR]
(SK047 (SK042 (BEXPR . (SREL (SEXPR . SLIT)(SEXPR . SVAR]
(SK048 (SK047 (SEXPR . SVAR]
(SK049 (SK011]
(SK050 (SK082]
```

```
(SK051 (SK070 (ARITH . %+%]
(SK052 (SK069]
(SK053 (SK051]
(SK054 (SK070 (ARITH . %-%]
(SK055 (REMST]
(SK056 (INT]
(SK057 (RND]
(SK058 (SQR]
(SK059 (STOPST]
(SK060 (DIMST (NEXPR . NOTONE]
(SK061 (FORNEXTST (FORST (NUMVAR . (NVAR . 1))
                         (NEXPR . NLIT)(NEXPR . NLIT)(STEPEXPR . NULL))
                  (NEXTST (NUMVAR . (NVAR . 1]
(SK062 (SK061 (NEXPR . NLIT)(NEXPR . NVAR]
(SK063 (SK061 (NEXPR . SIMNEXPR)(NEXPR . SIMNEXPR)
              (STEPEXPR . (STEP (NEXPR . NOTONE]
(SK064 (SK063 (STEPEXPR . (STEP (NEXPR . NEG]
(SK065 (SUBSVAR (NEXPR . NVAR]
(SK066 (SK001 (EXPRLIST . SUBSVAR]
(SK067 (SUBNVAR (NEXPR . NVAR]
(SK068 (SK001 (EXPRLIST . SUBNVAR]
(SK069 (SK012 (STRVAR . (SVAR . 1))
              (SEXPR . (CONCATSEXPR (SEXPR . (SVAR . 1]
(SK070 (SK011 (NUMVAR . (NVAR . 1))
              (NEXPR . (SIMARITH (SIMNEXPR . (NVAR . 1]
(SK071 (COMPLARITH]
(SK072 (SK028 (EXPR . SLIT)(EXPRLIST . CONCATSEXPR]
(SK075 (BOOLEREL (BOOLOP . %AND%]
(SK076 (SK075 (BOOLOP . %OR%]
(SK077 (SK075 (BOOLOP . %NOT%]
(SK078 (SK011]
(SK079 (SK082]
(SK080 (SK012]
(SK081 (SK083]
(SK082 (SK011 (NEXPR . NVAR]
(SK083 (SK012 (SEXPR . SVAR]
(SK084 (SK011 (NUMVAR . SUBNVAR) (NEXPR . NEXPR]
(SK085 (SK012 (STRVAR . SUBSVAR) (SEXPR . SEXPR]
(SK086 (SK015 (READST (VARLIST . SUBNVAR]
(SK087 (SK016 (READST (VARLIST . SUBSVAR]
(SK088 (SUBSTRINGSEXPR (SEXPR . STRVAR)
                       (SUBSTRING (NEXPR . NLIT) (NEXPR . NLIT]
(SK089 (SK088 (NEXPR . NVAR) (NEXPR . NVAR]
(SK090 (SK088 (NEXPR . NVAR]
(SK091 (LEN]
(SK092 (SK080 (STRVAR . STRVAR) (SEXPR . NULL]
(SK093 (SK028 (EXPRLIST . EXPRS]
```

Note: A right bracket matches all open left parentheses.

Appendix G. LISP Notation for the SKILLSNET

```
(SK001 A (SK002) H (SK002 SK028))
(SK002 H (SK003 SK028 SK005))
(SK003 S (SK004) A (SK004) H (SK028 SK068))
(SK004 H (SK005 SK066 SK030))
(SK005 H (SK029 SK008 SK006) A (SK008) P (SK001))
(SK006 H (SK007 SK009 SK029 SK035) P (SK003) A (SK009))
(SK007 H (SK029 SK035 SK039 SK068 SK010) P (SK001 SK003) A (SK010))
(SK008 H (SK009 SK074) P (SK002))
(SK009 H (SK010 SK034) P (SK004))
(SK010 H (SK034 SK074 SK039) P (SK002 SK004))
(SK011 A (SK012) H (SK012 SK033 SK049 SK082) P (SK001))
(SK012 H (SK080 SK083) P (SK002))
(SK013 A (SK014 SK015) H (SK015 SK022) S (SK014) P (SK011))
(SK014 H (SK032 SK023 SK024) P (SK012))
(SK015 H (SK016 SK017 SK086) A (SK016) P (SK011))
(SK016 H (SK026 SK018) P (SK012))
(SK017 H (SK018 SK020) A (SK018) P (SK015))
(SK018 H (SK019 SK021) P (SK016))
(SK019 P (SK017 SK018) A (SK017))
(SK020 A (SK021) H (SK021) P (SK017))
(SK021 P (SK018))
(SK022 S (SK023) A (SK013 SK023 SK025) H (SK025) P (SK013))
(SK023 H (SK024 SK026) A (SK014 SK026) P (SK014))
(SK024 H (SK027) A (SK013 SK027) P (SK013 SK014))
(SK025 S (SK026) A (SK015 SK026) P (SK015))
(SK026 A (SK016) H (SK027) P (SK016))
(SK027 P (SK015 SK016) A (SK015))
(SK028 A (SK002 SK030) S (SK030) P (SK003 SK002))
(SK029 H (SK074) A (SK002 SK074) P (SK005 SK002))
(SK030 A (SK002) H (SK029) P (SK004 SK002))
(SK031 S (SK032) A (SK032) P (SK014))
(SK032 P (SK013))
(SK033 P (SK044 SK082))
(SK034 H (SK069) P (SK008 SK012))
(SK035 H (SK034 SK051 SK070) A (SK034) P (SK005 SK011))
(SK036 P (SK011))
(SK038 H (SK041) P (SK002))
(SK039 H (SK038 SK040) A (SK038) P (SK001))
(SK040 H (SK041) A (SK041) P (SK001 SK003))
(SK041 P (SK002 SK004))
(SK042 H (SK044 SK046 SK047) A (SK047) P (SK036 SK040 SK003 SK013))
(SK043 H (SK047 SK075 SK061) P (SK036 SK040 SK003 SK005 SK013))
(SK044 H (SK045) A (SK045) P (SK049 SK042))
(SK045 H (SK047) P (SK046))
(SK046 H (SK043  SK048 SK045) A (SK048) P (SK003 SK036 SK039 SK042))
(SK047 H (SK075 SK061) P (SK038 SK004 SK036 SK014))
(SK048 H (SK047) P (SK004 SK036 SK038 SK047))
(SK049 H (SK080) A (SK080) P (SK011))
(SK050 P (SK082))
(SK051 H (SK053 SK054 SK069) A (SK069) P (SK006 SK035))
(SK052 P (SK069) H (SK085))
```

(SK053 A (SK052) P (SK051) H (SK052))
(SK054 H (SK070) P (SK006 SK035))
(SK056 H (SK058) P (SK035))
(SK057 P (SK035))
(SK058 H (SK057) P (SK035))
(SK059 P (SK042))
(SK061 H (SK062) P (SK042))
(SK062 H (SK063) P (SK042))
(SK063 H (SK064) P (SK061))
(SK064 P (SK061))
(SK065 A (SK067) S (SK067) P (SK044 SK012))
(SK066 A (SK068) S (SK068) P (SK004))
(SK067 P (SK044 SK011))
(SK068 P (SK003 SK044))
(SK069 H (SK052) P (SK009 SK034))
(SK070 P (SK006 SK035) H (SK084))
(SK071 P (SK035))
(SK074 P (SK008 SK002) A (SK002) H (SK093))
(SK075 H (SK076) P (SK039))
(SK076 H (SK077) P (SK039))
(SK077 P (SK039))
(SK079 A (SK081) S (SK081) H (SK050) P (SK082))
(SK080 P (SK012) H (SK092) A (SK092))
(SK081 H (SK050) A (SK050) P (SK083))
(SK082 H (SK035 SK079) A (SK083) S (SK083) P (SK011))
(SK083 H (SK034 SK081) P (SK012))
(SK084 A (SK085) S (SK085))
(SK086 A (SK087) H (SK087) P (SK011))
(SK087 P (SK012))
(SK088 P (SK034) H (SK090))
(SK089 P (SK034))
(SK090 P (SK034) H (SK089))
(SK091 H (SK088 SK056))
(SK092 P (SK012))
(SK093 P (SK028 SK030))

Definitions of relations

A	conceptually Analogous to ...
S	Similarly difficult to use or learn as ...
H	Harder skills to use or learn are ...
P	pedagogically Prerequisite skills are ...